

HTML 5.1 2nd Edition

W3C Recommendation 3 October 2017

**This version:**

<https://www.w3.org/TR/2017/REC-html51-20171003/>

Latest published version:

<https://www.w3.org/TR/html51/>

Latest version of HTML:

<https://www.w3.org/TR/html/>

Editor's Draft:

<https://rawgit.com/w3c/html/html5.1-2/single-page.html>

Previous Versions:

<https://www.w3.org/TR/2017/PR-html51-20170803/>

Editors:

[Steve Faulkner](#) (The Paciello Group)

[Arron Eicholz](#) (Microsoft)

[Travis Leithead](#) (Microsoft)

[Alex Danilo](#) (Google)

Former Editors:

[Erika Doyle Navara](#) (Microsoft)

[Theresa O'Connor](#) (Apple Inc.)

[Robin Berjon](#) (W3C)

Participate:

[File an issue](#) ([open issues](#))

Others:

[Single page version](#)

Errata for this document are recorded as Github issues.

The English version of this specification is the only normative version. Non-normative [translations](#) may also be available.

[Copyright](#) © 2017 W3C® ([MIT](#), [ERCIM](#), [Keio](#), [Beihang](#)). W3C liability, trademark and document use rules apply.

Abstract

This specification defines the 2nd edition of the 5th major version, first minor revision of the core language of the World Wide Web: the Hypertext Markup Language (HTML). In this version, a significant error with [HTML 5.1](#) has been rectified, as noted in the [Errata corrected](#) section. Meanwhile, new features continue to be introduced to help Web application authors, new elements continue to be introduced based on research into prevailing authoring practices, and special attention continues to be given to defining clear conformance criteria for user agents in an effort to improve interoperability in the development of [HTML 5.2](#).

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the [W3C technical reports index](#) at <https://www.w3.org/TR/>.

This specification defines the 2nd edition of the 5th major version, first minor revision of the core language of the World Wide Web: the Hypertext Markup Language (HTML). In this version, a significant error with [HTML 5.1](#) has been rectified, as noted in the [Errata corrected](#) section.

[Errata](#) for this document are recorded as Github issues with the "errata" label. The [latest HTML 5.1 editors' draft](#) shows proposed resolutions of errata *in situ*.

All interested parties are invited to provide implementation and bug reports and other comments through the Working Group's [Github repository](#). These will generally be considered in the development of [HTML 5.2](#).

The [implementation report](#) produced for HTML 5.1 demonstrates that in almost every case changes are matched by interoperable implementation.

The Working Group aims to produce an HTML 5.2 Recommendation in late 2017 that would obsolete this Recommendation.

This document was published by the [Web Platform Working Group](#) as a Recommendation. Feedback and comments on this specification are welcome. Please use [Github issues](#). Historical discussions can be found in the public-html@w3.org [archives](#).

This document has been reviewed by W3C Members, by software developers, and by other W3C groups and interested parties, and is endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document was produced by a group operating under the [5 February 2004 W3C Patent Policy](#). W3C maintains a [public list of any patent disclosures](#) made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains [Essential Claim\(s\)](#) must disclose the information in accordance with [section 6 of the W3C Patent Policy](#).

This document is governed by the [1 March 2017 W3C Process Document](#).

Table of Contents

1	Introduction
1.1	Background
1.2	Audience
1.3	Scope
1.4	History
1.5	Design notes
1.5.1	Serializability of script execution
1.5.2	Compliance with other specifications
1.5.3	Extensibility
1.6	HTML vs XHTML
1.7	Structure of this specification
1.7.1	How to read this specification
1.7.2	Typographic conventions
1.8	Privacy concerns
1.9	A quick introduction to HTML
1.9.1	Writing secure applications with HTML
1.9.2	Common pitfalls to avoid when using the scripting APIs
1.9.3	How to catch mistakes when writing HTML: validators and conformance checkers
1.10	Conformance requirements for authors
1.10.1	Presentational markup
1.10.2	Syntax errors
1.10.3	Restrictions on content models and on attribute values
1.11	Suggested reading
2	Common infrastructure

2.1	Terminology
2.1.1	Resources
2.1.2	XML
2.1.3	DOM trees
2.1.4	Scripting
2.1.5	Plugin Content Handlers
2.1.6	Character encodings
2.2	Conformance requirements
2.2.1	Conformance classes
2.2.2	Dependencies
2.2.3	Extensibility
2.2.4	Interactions with XPath and XSLT
2.3	Case-sensitivity and string comparison
2.4	Common microsyntaxes
2.4.1	Common parser idioms
2.4.2	Boolean attributes
2.4.3	Keywords and enumerated attributes
2.4.4	Numbers
2.4.4.1	Signed integers
2.4.4.2	Non-negative integers
2.4.4.3	Floating-point numbers
2.4.4.4	Percentages and lengths
2.4.4.5	Non-zero percentages and lengths
2.4.4.6	Lists of floating-point numbers
2.4.4.7	Lists of dimensions
2.4.5	Dates and times
2.4.5.1	Months
2.4.5.2	Dates
2.4.5.3	Yearless dates
2.4.5.4	Times
2.4.5.5	Floating dates and times
2.4.5.6	Time zones
2.4.5.7	Global dates and times
2.4.5.8	Weeks
2.4.5.9	Durations
2.4.5.10	Vaguer moments in time
2.4.6	Colors
2.4.7	Space-separated tokens
2.4.8	Comma-separated tokens
2.4.9	References
2.4.10	Media queries
2.5	URLs
2.5.1	Terminology
2.5.2	Parsing URLs
2.5.3	Dynamic changes to base URLs
2.6	Fetching resources
2.6.1	Terminology
2.6.2	Processing model
2.6.3	Encrypted HTTP and related security concerns
2.6.4	Determining the type of a resource
2.6.5	Extracting character encodings from meta elements
2.6.6	CORS settings attributes
2.7	Common DOM interfaces
2.7.1	Reflecting content attributes in IDL attributes
2.7.2	Collections
2.7.2.1	The HTMLA11Collection interface

2.7.2.2	The <code>HTMLFormControlsCollection</code> interface
2.7.2.3	The <code>HTMLOptionsCollection</code> interface
2.7.3	The <code>DOMStringMap</code> interface
2.7.4	The <code>DOMElementMap</code> interface
2.7.5	Garbage collection
2.8	Namespaces
2.9	Safe passing of structured data
2.9.1	Cloneable objects
2.9.2	Transferable objects
2.9.3	<code>StructuredCloneWithTransfer</code> (<i>input</i> , <i>transferList</i> , <i>targetRealm</i>)
2.9.4	<code>StructuredClone</code> (<i>input</i> , <i>targetRealm</i> [, <i>memory</i>])
2.9.5	<code>IsTransferable</code> (<i>O</i>)
2.9.6	<code>TransferHelper</code> (<i>input</i> , <i>targetRealm</i>)
3	Semantics, structure, and APIs of HTML documents
3.1	Documents
3.1.1	The Document object
3.1.2	Resource metadata management
3.1.3	DOM tree accessors
3.1.4	Loading XML documents
3.2	Elements
3.2.1	Semantics
3.2.2	Elements in the DOM
3.2.3	Element definitions
3.2.3.1	Attributes
3.2.4	Content models
3.2.4.1	The "nothing" content model
3.2.4.2	Kinds of content
3.2.4.2.1	Metadata content
3.2.4.2.2	Flow content
3.2.4.2.3	Sectioning content
3.2.4.2.4	Heading content
3.2.4.2.5	Phrasing content
3.2.4.2.6	Embedded content
3.2.4.2.7	Interactive content
3.2.4.2.8	Palpable content
3.2.4.2.9	Script-supporting elements
3.2.4.3	Transparent content models
3.2.4.4	Paragraphs
3.2.5	Global attributes
3.2.5.1	The <code>id</code> attribute
3.2.5.2	The <code>title</code> attribute
3.2.5.3	The <code>lang</code> and <code>xml:lang</code> attributes
3.2.5.4	The <code>translate</code> attribute
3.2.5.5	The <code>xml:base</code> attribute (XML only)
3.2.5.6	The <code>dir</code> attribute
3.2.5.7	The <code>class</code> attribute
3.2.5.8	The <code>style</code> attribute
3.2.5.9	Embedding custom non-visible data with the <code>data-*</code> attributes
3.2.6	Requirements relating to the bidirectional algorithm
3.2.6.1	Authoring conformance criteria for bidirectional-algorithm formatting characters
3.2.6.2	User agent conformance criteria
3.2.7	WAI-ARIA and HTML Accessibility API Mappings
3.2.7.1	ARIA Authoring Requirements
3.2.7.2	Conformance Checker Implementation Requirements
3.2.7.3	User Agent Implementation Requirements

- 3.2.7.3.1 ARIA Role Attribute
- 3.2.7.3.2 State and Property Attributes
- 3.2.7.4 Allowed ARIA roles, states and properties

4 The elements of HTML

- 4.1 The root element
 - 4.1.1 The `html` element
- 4.2 Document metadata
 - 4.2.1 The `head` element
 - 4.2.2 The `title` element
 - 4.2.3 The `base` element
 - 4.2.4 The `link` element
 - 4.2.5 The `meta` element
 - 4.2.5.1 Standard metadata names
 - 4.2.5.2 Other metadata names
 - 4.2.5.3 Pragma directives
 - 4.2.5.4 Other pragma directives
 - 4.2.5.5 Specifying the document's character encoding
 - 4.2.6 The `style` element
 - 4.2.7 Interactions of styling and scripting
- 4.3 Sections
 - 4.3.1 The `body` element
 - 4.3.2 The `article` element
 - 4.3.3 The `section` element
 - 4.3.4 The `nav` element
 - 4.3.5 The `aside` element
 - 4.3.6 The `h1`, `h2`, `h3`, `h4`, `h5`, and `h6` elements
 - 4.3.7 The `header` element
 - 4.3.8 The `footer` element
 - 4.3.9 The `address` element
 - 4.3.10 Headings and sections
 - 4.3.10.1 Creating an outline
 - 4.3.11 Usage summary
 - 4.3.11.1 Article or section?
- 4.4 Grouping content
 - 4.4.1 The `p` element
 - 4.4.2 The `hr` element
 - 4.4.3 The `pre` element
 - 4.4.4 The `blockquote` element
 - 4.4.5 The `ol` element
 - 4.4.6 The `ul` element
 - 4.4.7 The `li` element
 - 4.4.8 The `dl` element
 - 4.4.9 The `dt` element
 - 4.4.10 The `dd` element
 - 4.4.11 The `figure` element
 - 4.4.12 The `figcaption` element
 - 4.4.13 The `main` element
 - 4.4.14 The `div` element
- 4.5 Text-level semantics
 - 4.5.1 The `a` element
 - 4.5.2 The `em` element
 - 4.5.3 The `strong` element
 - 4.5.4 The `small` element
 - 4.5.5 The `s` element
 - 4.5.6 The `cite` element

4.5.7	The <code>q</code> element
4.5.8	The <code>dfn</code> element
4.5.9	The <code>abbr</code> element
4.5.10	The <code>ruby</code> element
4.5.11	The <code>rb</code> element
4.5.12	The <code>rt</code> element
4.5.13	The <code>rtc</code> element
4.5.14	The <code>rp</code> element
4.5.15	The <code>data</code> element
4.5.16	The <code>time</code> element
4.5.17	The <code>code</code> element
4.5.18	The <code>var</code> element
4.5.19	The <code>samp</code> element
4.5.20	The <code>kbd</code> element
4.5.21	The <code>sub</code> and <code>sup</code> elements
4.5.22	The <code>i</code> element
4.5.23	The <code>b</code> element
4.5.24	The <code>u</code> element
4.5.25	The <code>mark</code> element
4.5.26	The <code>bdi</code> element
4.5.27	The <code>bdo</code> element
4.5.28	The <code>span</code> element
4.5.29	The <code>br</code> element
4.5.30	The <code>wbr</code> element
4.5.31	Usage summary
4.6	Edits
4.6.1	The <code>ins</code> element
4.6.2	The <code>del</code> element
4.6.3	Attributes common to <code>ins</code> and <code>del</code> elements
4.6.4	Edits and paragraphs
4.6.5	Edits and lists
4.6.6	Edits and tables
4.7	Embedded content
4.7.1	Introduction
4.7.2	Dependencies
4.7.3	The <code>picture</code> element
4.7.4	The <code>source</code> element when used with the <code>picture</code> element
4.7.5	The <code>img</code> element
4.7.5.1	Requirements for providing text to act as an alternative for images
4.7.5.1.1	Examples of scenarios where users benefit from text alternatives for images
4.7.5.1.2	General guidelines
4.7.5.1.3	A link or button containing nothing but an image
4.7.5.1.4	Graphical Representations: Charts, diagrams, graphs, maps, illustrations
4.7.5.1.5	Images of text
4.7.5.1.6	Images that include text
4.7.5.1.7	Images that enhance the themes or subject matter of the page content
4.7.5.1.8	A graphical representation of some of the surrounding text
4.7.5.1.9	A purely decorative image that doesn't add any information
4.7.5.1.10	Inline images
4.7.5.1.11	A group of images that form a single larger picture with no links
4.7.5.1.12	Image maps
4.7.5.1.13	A group of images that form a single larger picture with links
4.7.5.1.14	Images of Pictures
4.7.5.1.15	Webcam images
4.7.5.1.16	When a text alternative is not available at the time of publication
4.7.5.1.17	An image not intended for the user

4.7.5.1.18	Icon Images
4.7.5.1.19	Logos, insignia, flags, or emblems
4.7.5.1.20	CAPTCHA Images
4.7.5.1.21	An image in a picture element
4.7.5.1.22	Guidance for markup generators
4.7.5.1.23	Guidance for conformance checkers
4.7.6	The <code>iframe</code> element
4.7.7	The <code>embed</code> element
4.7.8	The <code>object</code> element
4.7.9	The <code>param</code> element
4.7.10	The <code>video</code> element
4.7.11	The <code>audio</code> element
4.7.12	The <code>source</code> element
4.7.13	The <code>track</code> element
4.7.14	Media elements
4.7.14.1	Error codes
4.7.14.2	Location of the media resource
4.7.14.3	MIME types
4.7.14.4	Network states
4.7.14.5	Loading the media resource
4.7.14.6	Offsets into the media resource
4.7.14.7	Ready states
4.7.14.8	Playing the media resource
4.7.14.9	Seeking
4.7.14.10	Media resources with multiple media tracks
4.7.14.10.1	<code>AudioTrackList</code> and <code>VideoTrackList</code> objects
4.7.14.10.2	Selecting specific audio and video tracks declaratively
4.7.14.11	Timed text tracks
4.7.14.11.1	Text track model
4.7.14.11.2	Sourcing in-band text tracks
4.7.14.11.3	Sourcing out-of-band text tracks
4.7.14.11.4	Guidelines for exposing cues in various formats as text track cues
4.7.14.11.5	Text track API
4.7.14.11.6	Text tracks exposing in-band metadata
4.7.14.11.7	Text tracks describing chapters
4.7.14.11.8	Event handlers for objects of the text track APIs
4.7.14.11.9	Best practices for metadata text tracks
4.7.14.12	User interface
4.7.14.13	Time ranges
4.7.14.14	The <code>TrackEvent</code> interface
4.7.14.15	Event summary
4.7.14.16	Security and privacy considerations
4.7.14.17	Best practices for authors using media elements
4.7.14.18	Best practices for implementors of media elements
4.7.15	The <code>map</code> element
4.7.16	The <code>area</code> element
4.7.17	Image maps
4.7.17.1	Authoring
4.7.17.2	Processing model
4.7.18	MathML
4.7.19	SVG
4.7.20	Dimension attributes
4.8	Links
4.8.1	Introduction
4.8.2	Links created by <code>a</code> and <code>area</code> elements
4.8.3	API for <code>a</code> and <code>area</code> elements

4.8.4	Following hyperlinks
4.8.5	Downloading resources
4.8.6	Link types
4.8.6.1	Link type "alternate"
4.8.6.2	Link type "author"
4.8.6.3	Link type "bookmark"
4.8.6.4	Link type "help"
4.8.6.5	Link type "icon"
4.8.6.6	Link type "license"
4.8.6.7	Link type "nofollow"
4.8.6.8	Link type "noreferrer"
4.8.6.9	Link type "search"
4.8.6.10	Link type "stylesheet"
4.8.6.11	Link type "tag"
4.8.6.12	Sequential link types
4.8.6.12.1	Link type "next"
4.8.6.12.2	Link type "prev"
4.8.6.13	Other link types
4.9	Tabular data
4.9.1	The <code>table</code> element
4.9.1.1	Techniques for describing tables
4.9.1.2	Techniques for table design
4.9.2	The <code>caption</code> element
4.9.3	The <code>colgroup</code> element
4.9.4	The <code>col</code> element
4.9.5	The <code>tbody</code> element
4.9.6	The <code>thead</code> element
4.9.7	The <code>tfoot</code> element
4.9.8	The <code>tr</code> element
4.9.9	The <code>td</code> element
4.9.10	The <code>th</code> element
4.9.11	Attributes common to <code>td</code> and <code>th</code> elements
4.9.12	Processing model
4.9.12.1	Forming a table
4.9.12.2	Forming relationships between data cells and header cells
4.9.13	Examples
4.10	Forms
4.10.1	Introduction
4.10.1.1	Writing a form's user interface
4.10.1.2	Implementing the server-side processing for a form
4.10.1.3	Configuring a form to communicate with a server
4.10.1.4	Client-side form validation
4.10.1.5	Enabling client-side automatic filling of form controls
4.10.1.6	Improving the user experience on mobile devices
4.10.1.7	The difference between the field type, the autofill field name, and the input modality
4.10.1.8	Date, time, and number formats
4.10.2	Categories
4.10.3	The <code>form</code> element
4.10.4	The <code>label</code> element
4.10.5	The <code>input</code> element
4.10.5.1	States of the <code>type</code> attribute
4.10.5.1.1	Hidden state (<code>type=hidden</code>)
4.10.5.1.2	Text (<code>type=text</code>) state and Search state (<code>type=search</code>)
4.10.5.1.3	Telephone state (<code>type=tel</code>)
4.10.5.1.4	URL state (<code>type=url</code>)
4.10.5.1.5	E-mail state (<code>type=email</code>)

4.10.5.1.6	Password state (<code>type=password</code>)
4.10.5.1.7	Date state (<code>type=date</code>)
4.10.5.1.8	Month state (<code>type=month</code>)
4.10.5.1.9	Week state (<code>type=week</code>)
4.10.5.1.10	Time state (<code>type=time</code>)
4.10.5.1.11	Local Date and Time state (<code>type=datetime-local</code>)
4.10.5.1.12	Number state (<code>type=number</code>)
4.10.5.1.13	Range state (<code>type=range</code>)
4.10.5.1.14	Color state (<code>type=color</code>)
4.10.5.1.15	Checkbox state (<code>type=checkbox</code>)
4.10.5.1.16	Radio Button state (<code>type=radio</code>)
4.10.5.1.17	File Upload state (<code>type=file</code>)
4.10.5.1.18	Submit Button state (<code>type=submit</code>)
4.10.5.1.19	Image Button state (<code>type=image</code>)
4.10.5.1.20	Reset Button state (<code>type=reset</code>)
4.10.5.1.21	Button state (<code>type=button</code>)
4.10.5.2	Implementation notes regarding localization of form controls
4.10.5.3	Common input element attributes
4.10.5.3.1	The <code>maxlength</code> and <code>minlength</code> attributes
4.10.5.3.2	The <code>size</code> attribute
4.10.5.3.3	The <code>readonly</code> attribute
4.10.5.3.4	The <code>required</code> attribute
4.10.5.3.5	The <code>multiple</code> attribute
4.10.5.3.6	The <code>pattern</code> attribute
4.10.5.3.7	The <code>min</code> and <code>max</code> attributes
4.10.5.3.8	The <code>step</code> attribute
4.10.5.3.9	The <code>list</code> attribute
4.10.5.3.10	The <code>placeholder</code> attribute
4.10.5.4	Common input element APIs
4.10.5.5	Common event behaviors
4.10.6	The <code>button</code> element
4.10.7	The <code>select</code> element
4.10.8	The <code>datalist</code> element
4.10.9	The <code>optgroup</code> element
4.10.10	The <code>option</code> element
4.10.11	The <code>textarea</code> element
4.10.12	The <code>keygen</code> element
4.10.13	The <code>output</code> element
4.10.14	The <code>progress</code> element
4.10.15	The <code>meter</code> element
4.10.16	The <code>fieldset</code> element
4.10.17	The <code>legend</code> element
4.10.18	Form control infrastructure
4.10.18.1	A form control value
4.10.18.2	Mutability
4.10.18.3	Association of controls and forms
4.10.19	Attributes common to form controls
4.10.19.1	Naming form controls: the <code>name</code> attribute
4.10.19.2	Submitting element directionality: the <code>dirname</code> attribute
4.10.19.3	Limiting user input length: the <code>maxlength</code> attribute
4.10.19.4	Setting minimum input length requirements: the <code>minlength</code> attribute
4.10.19.5	Enabling and disabling form controls: the <code>disabled</code> attribute
4.10.19.6	Form submission
4.10.19.6.1	Autofocusing a form control: the <code>autofocus</code> attribute
4.10.19.7	Input modalities: the <code>inputmode</code> attribute

4.10.19.8	Autofill
4.10.19.8.1	Autofilling form controls: the <code>autocomplete</code> attribute
4.10.19.8.2	Processing model
4.10.20	APIs for text field selections
4.10.21	Constraints
4.10.21.1	Definitions
4.10.21.2	Constraint validation
4.10.21.3	The constraint validation API
4.10.21.4	Security
4.10.22	Form submission
4.10.22.1	Introduction
4.10.22.2	Implicit submission
4.10.22.3	Form submission algorithm
4.10.22.4	Constructing the form data set
4.10.22.5	Selecting a form submission encoding
4.10.22.6	URL-encoded form data
4.10.22.7	Multipart form data
4.10.22.8	Plain text form data
4.10.23	Resetting a form
4.11	Interactive elements
4.11.1	The <code>details</code> element
4.11.2	The <code>summary</code> element
4.11.3	The <code>menu</code> element
4.11.4	The <code>menuitem</code> element
4.11.5	Context menus
4.11.5.1	Declaring a context menu
4.11.5.2	Processing model
4.11.5.3	The <code>RelatedEvent</code> interfaces
4.11.6	Commands
4.11.6.1	Facets
4.11.6.2	Using the <code>a</code> element to define a command
4.11.6.3	Using the <code>button</code> element to define a command
4.11.6.4	Using the <code>input</code> element to define a command
4.11.6.5	Using the <code>option</code> element to define a command
4.11.6.6	Using the <code>menuitem</code> element to define a command
4.11.6.7	Using the <code>accesskey</code> attribute on a <code>label</code> element to define a command
4.11.6.8	Using the <code>accesskey</code> attribute on a <code>legend</code> element to define a command
4.11.6.9	Using the <code>accesskey</code> attribute to define a command on other elements
4.12	Scripting
4.12.1	The <code>script</code> element
4.12.1.1	Processing model
4.12.1.2	Scripting languages
4.12.1.3	Restrictions for contents of <code>script</code> elements
4.12.1.4	Inline documentation for external scripts
4.12.1.5	Interaction of <code>script</code> elements and XSLT
4.12.2	The <code>noscript</code> element
4.12.3	The <code>template</code> element
4.12.3.1	Interaction of <code>template</code> elements with XSLT and XPath
4.12.4	The <code>canvas</code> element
4.12.4.1	Color spaces and color correction
4.12.4.2	Serializing bitmaps to a file
4.12.4.3	Security with <code>canvas</code> elements
4.13	Common idioms without dedicated elements
4.13.1	Subheadings, subtitles, alternative titles and taglines
4.13.2	Bread crumb navigation
4.13.3	Tag clouds

4.13.4	Conversations
4.13.5	Footnotes
4.14	Disabled elements
4.15	Matching HTML elements using selectors
4.15.1	Case-sensitivity
4.15.2	Pseudo-classes
5	User interaction
5.1	The hidden attribute
5.2	Inert subtrees
5.3	Activation
5.4	Focus
5.4.1	Introduction
5.4.2	Data model
5.4.3	The <code>tabindex</code> attribute
5.4.4	Processing model
5.4.5	Sequential focus navigation
5.4.6	Focus management APIs
5.5	Assigning keyboard shortcuts
5.5.1	Introduction
5.5.2	The <code>accesskey</code> attribute
5.5.3	Processing model
5.6	Editing
5.6.1	Making document regions editable: The <code>contenteditable</code> content attribute
5.6.2	Making entire documents editable: The <code>designMode</code> IDL attribute
5.6.3	Best practices for in-page editors
5.6.4	Editing APIs
5.6.5	Spelling and grammar checking
5.7	Drag and drop
5.7.1	Introduction
5.7.2	The drag data store
5.7.3	The <code>DataTransfer</code> interface
5.7.3.1	The <code>DataTransferItemList</code> interface
5.7.3.2	The <code>DataTransferItem</code> interface
5.7.4	The <code>DragEvent</code> interface
5.7.5	Drag-and-drop processing model
5.7.6	Events summary
5.7.7	The <code>draggable</code> attribute
5.7.8	The <code>dropzone</code> attribute
5.7.9	Security risks in the drag-and-drop model
6	Loading Web pages
6.1	Browsing contexts
6.1.1	Nested browsing contexts
6.1.1.1	Navigating nested browsing contexts in the DOM
6.1.2	Auxiliary browsing contexts
6.1.2.1	Navigating auxiliary browsing contexts in the DOM
6.1.3	Security
6.1.4	Groupings of browsing contexts
6.1.5	Browsing context names
6.1.6	Script settings for browsing contexts
6.2	Security infrastructure for <code>Window</code> , <code>WindowProxy</code> , and <code>Location</code> objects
6.2.1	Integration with IDL
6.2.2	Shared internal slot: <code>[[CrossOriginPropertyDescriptorMap]]</code>
6.2.3	Shared abstract operations
6.2.3.1	<code>CrossOriginProperties</code> (<i>O</i>)
6.2.3.2	<code>IsPlatformObjectSameOrigin</code> (<i>O</i>)

6.2.3.3	CrossOriginGetOwnPropertyHelper (<i>O</i> , <i>P</i>)
6.2.3.3.1	CrossOriginPropertyDescriptor (<i>crossOriginProperty</i> , <i>originalDesc</i>)
6.2.3.3.2	CrossOriginFunctionWrapper (<i>needsWrapping</i> , <i>functionToWrap</i>)
6.2.3.4	CrossOriginGet (<i>O</i> , <i>P</i> , <i>Receiver</i>)
6.2.3.5	CrossOriginSet (<i>O</i> , <i>P</i> , <i>V</i> , <i>Receiver</i>)
6.2.3.6	CrossOriginOwnPropertyKeys (<i>O</i>)
6.3	The window object
6.3.1	APIs for creating and navigating browsing contexts by name
6.3.2	Accessing other browsing contexts
6.3.3	Named access on the window object
6.3.4	Garbage collection and browsing contexts
6.3.5	Closing browsing contexts
6.3.6	Browser interface elements
6.3.7	The WindowProxy object
6.3.7.1	The WindowProxy internal methods
6.3.7.1.1	[[GetPrototypeOf]] ()
6.3.7.1.2	[[SetPrototypeOf]] (<i>V</i>)
6.3.7.1.3	[[IsExtensible]] ()
6.3.7.1.4	[[PreventExtensions]] ()
6.3.7.1.5	[[GetOwnProperty]] (<i>P</i>)
6.3.7.1.6	[[DefineOwnProperty]] (<i>P</i> , <i>Desc</i>)
6.3.7.1.7	[[Get]] (<i>P</i> , <i>Receiver</i>)
6.3.7.1.8	[[Set]] (<i>P</i> , <i>V</i> , <i>Receiver</i>)
6.3.7.1.9	[[Delete]] (<i>P</i>)
6.3.7.1.10	[[OwnPropertyKeys]] ()
6.4	Origin
6.4.1	Relaxing the same-origin restriction
6.5	Sandboxing
6.6	Session history and navigation
6.6.1	The session history of browsing contexts
6.6.2	The History interface
6.6.3	Implementation notes for session history
6.6.4	The Location interface
6.7	Browsing the Web
6.7.1	Navigating across documents
6.7.2	Page load processing model for HTML files
6.7.3	Page load processing model for XML files
6.7.4	Page load processing model for text files
6.7.5	Page load processing model for multipart/x-mixed-replace resources
6.7.6	Page load processing model for media
6.7.7	Page load processing model for content that uses plugins
6.7.8	Page load processing model for inline content that doesn't have a DOM
6.7.9	Navigating to a fragment identifier
6.7.10	History traversal
6.7.10.1	Persisted user state restoration
6.7.10.2	The PopStateEvent interface
6.7.10.3	The HashChangeEvent interface
6.7.10.4	The PageTransitionEvent interface
6.7.11	Unloading documents
6.7.11.1	The BeforeUnloadEvent interface
6.7.12	Aborting a document load
6.7.13	Browser state
7	Web application APIs
7.1	Scripting
7.1.1	Introduction

7.1.2	Enabling and disabling scripting
7.1.3	Processing model
7.1.3.1	Definitions
7.1.3.2	Fetching scripts
7.1.3.3	Creating scripts
7.1.3.4	Calling scripts
7.1.3.5	Realms, settings objects, and global objects
7.1.3.5.1	Entry
7.1.3.5.2	Incumbent
7.1.3.5.3	Current
7.1.3.5.4	Relevant
7.1.3.6	Killing scripts
7.1.3.7	Integration with the JavaScript job queue
7.1.3.7.1	<code>EnqueueJob(<i>queueName</i> , <i>job</i> , <i>arguments</i>)</code>
7.1.3.8	Runtime script errors
7.1.3.8.1	Runtime script errors in documents
7.1.3.8.2	The <code>ErrorEvent</code> interface
7.1.3.9	Unhandled promise rejections
7.1.3.9.1	The <code>HostPromiseRejectionTracker</code> implementation
7.1.3.9.2	The <code>PromiseRejectionEvent</code> interface
7.1.3.10	<code>HostEnsureCanCompileStrings(<i>callerRealm</i> , <i>calleeRealm</i>)</code>
7.1.4	Event loops
7.1.4.1	Definitions
7.1.4.2	Processing model
7.1.4.3	Generic task sources
7.1.5	Events
7.1.5.1	Event handlers
7.1.5.2	Event handlers on elements, Document objects, and Window objects
7.1.5.2.1	IDL definitions
7.1.5.3	Event firing
7.1.5.4	Events and the window object
7.2	Base64 utility methods
7.3	Dynamic markup insertion
7.3.1	Opening the input stream
7.3.2	Closing the input stream
7.3.3	<code>document.write()</code>
7.3.4	<code>document.writeln()</code>
7.4	Timers
7.5	User prompts
7.5.1	Simple dialogs
7.5.2	Printing
7.5.3	Dialogs implemented using separate documents with <code>showModalDialog()</code>
7.6	System state and capabilities
7.6.1	The Navigator object
7.6.1.1	Client identification
7.6.1.2	Language preferences
7.6.1.3	Custom scheme handler: the <code>registerProtocolHandler()</code> method
7.6.1.3.1	Security and privacy
7.6.1.4	Cookies
7.6.1.5	Plugins
7.7	Images
7.8	Animation Frames
8	The HTML syntax
8.1	Writing HTML documents
8.1.1	The DOCTYPE

8.1.2	Elements
8.1.2.1	Start tags
8.1.2.2	End tags
8.1.2.3	Attributes
8.1.2.4	Optional tags
8.1.2.5	Restrictions on content models
8.1.2.6	Restrictions on the contents of raw text and escapable raw text elements
8.1.3	Text
8.1.3.1	Newlines
8.1.4	Character references
8.1.5	CDATA sections
8.1.6	Comments
8.2	Parsing HTML documents
8.2.1	Overview of the parsing model
8.2.2	The input byte stream
8.2.2.1	Parsing with a known character encoding
8.2.2.2	Determining the character encoding
8.2.2.3	Character encodings
8.2.2.4	Changing the encoding while parsing
8.2.2.5	Preprocessing the input stream
8.2.3	Parse state
8.2.3.1	The insertion mode
8.2.3.2	The stack of open elements
8.2.3.3	The list of active formatting elements
8.2.3.4	The element pointers
8.2.3.5	Other parsing state flags
8.2.4	Tokenization
8.2.4.1	Data state
8.2.4.2	Character reference in data state
8.2.4.3	RCDATA state
8.2.4.4	Character reference in RCDATA state
8.2.4.5	RAWTEXT state
8.2.4.6	Script data state
8.2.4.7	PLAINTEXT state
8.2.4.8	Tag open state
8.2.4.9	End tag open state
8.2.4.10	Tag name state
8.2.4.11	RCDATA less-than sign state
8.2.4.12	RCDATA end tag open state
8.2.4.13	RCDATA end tag name state
8.2.4.14	RAWTEXT less-than sign state
8.2.4.15	RAWTEXT end tag open state
8.2.4.16	RAWTEXT end tag name state
8.2.4.17	Script data less-than sign state
8.2.4.18	Script data end tag open state
8.2.4.19	Script data end tag name state
8.2.4.20	Script data escape start state
8.2.4.21	Script data escape start dash state
8.2.4.22	Script data escaped state
8.2.4.23	Script data escaped dash state
8.2.4.24	Script data escaped dash dash state
8.2.4.25	Script data escaped less-than sign state
8.2.4.26	Script data escaped end tag open state
8.2.4.27	Script data escaped end tag name state
8.2.4.28	Script data double escape start state
8.2.4.29	Script data double escaped state

8.2.4.30	Script data double escaped dash state
8.2.4.31	Script data double escaped dash dash state
8.2.4.32	Script data double escaped less-than sign state
8.2.4.33	Script data double escape end state
8.2.4.34	Before attribute name state
8.2.4.35	Attribute name state
8.2.4.36	After attribute name state
8.2.4.37	Before attribute value state
8.2.4.38	Attribute value (double-quoted) state
8.2.4.39	Attribute value (single-quoted) state
8.2.4.40	Attribute value (unquoted) state
8.2.4.41	Character reference in attribute value state
8.2.4.42	After attribute value (quoted) state
8.2.4.43	Self-closing start tag state
8.2.4.44	Bogus comment state
8.2.4.45	Markup declaration open state
8.2.4.46	Comment start state
8.2.4.47	Comment start dash state
8.2.4.48	Comment state
8.2.4.49	Comment end dash state
8.2.4.50	Comment end state
8.2.4.51	Comment end bang state
8.2.4.52	DOCTYPE state
8.2.4.53	Before DOCTYPE name state
8.2.4.54	DOCTYPE name state
8.2.4.55	After DOCTYPE name state
8.2.4.56	After DOCTYPE public keyword state
8.2.4.57	Before DOCTYPE public identifier state
8.2.4.58	DOCTYPE public identifier (double-quoted) state
8.2.4.59	DOCTYPE public identifier (single-quoted) state
8.2.4.60	After DOCTYPE public identifier state
8.2.4.61	Between DOCTYPE public and system identifiers state
8.2.4.62	After DOCTYPE system keyword state
8.2.4.63	Before DOCTYPE system identifier state
8.2.4.64	DOCTYPE system identifier (double-quoted) state
8.2.4.65	DOCTYPE system identifier (single-quoted) state
8.2.4.66	After DOCTYPE system identifier state
8.2.4.67	Bogus DOCTYPE state
8.2.4.68	CDATA section state
8.2.4.69	Tokenizing character references
8.2.5	Tree construction
8.2.5.1	Creating and inserting nodes
8.2.5.2	Parsing elements that contain only text
8.2.5.3	Closing elements that have implied end tags
8.2.5.4	The rules for parsing tokens in HTML content
8.2.5.4.1	The "initial" insertion mode
8.2.5.4.2	The "before html" insertion mode
8.2.5.4.3	The "before head" insertion mode
8.2.5.4.4	The "in head" insertion mode
8.2.5.4.5	The "in head noscript" insertion mode
8.2.5.4.6	The "after head" insertion mode
8.2.5.4.7	The "in body" insertion mode
8.2.5.4.8	The "text" insertion mode
8.2.5.4.9	The "in table" insertion mode
8.2.5.4.10	The "in table text" insertion mode
8.2.5.4.11	The "in caption" insertion mode

8.2.5.4.12	The "in column group" insertion mode
8.2.5.4.13	The "in table body" insertion mode
8.2.5.4.14	The "in row" insertion mode
8.2.5.4.15	The "in cell" insertion mode
8.2.5.4.16	The "in select" insertion mode
8.2.5.4.17	The "in select in table" insertion mode
8.2.5.4.18	The "in template" insertion mode
8.2.5.4.19	The "after body" insertion mode
8.2.5.4.20	The "in frameset" insertion mode
8.2.5.4.21	The "after frameset" insertion mode
8.2.5.4.22	The "after after body" insertion mode
8.2.5.4.23	The "after after frameset" insertion mode
8.2.5.5	The rules for parsing tokens in foreign content
8.2.6	The end
8.2.7	Coercing an HTML DOM into an infoset
8.2.8	An introduction to error handling and strange cases in the parser
8.2.8.1	Misnested tags: <code><i></i></code>
8.2.8.2	Misnested tags: <code><p></p></code>
8.2.8.3	Unexpected markup in tables
8.2.8.4	Scripts that modify the page as it is being parsed
8.2.8.5	The execution of scripts that are moving across multiple documents
8.2.8.6	Unclosed formatting elements
8.3	Serializing HTML fragments
8.4	Parsing HTML fragments
8.5	Named character references
9	The XHTML syntax
9.1	Writing XHTML documents
9.2	Parsing XHTML documents
9.3	Serializing XHTML fragments
9.4	Parsing XHTML fragments
10	Rendering
10.1	Introduction
10.2	The CSS user agent style sheet and presentational hints
10.3	Non-replaced elements
10.3.1	Hidden elements
10.3.2	The page
10.3.3	Flow content
10.3.4	Phrasing content
10.3.5	Bidirectional text
10.3.6	Quotes
10.3.7	Sections and headings
10.3.8	Lists
10.3.9	Tables
10.3.10	Margin collapsing quirks
10.3.11	Form controls
10.3.12	The <code>hr</code> element
10.3.13	The <code>fieldset</code> and <code>legend</code> elements
10.4	Replaced elements
10.4.1	Embedded content
10.4.2	Images
10.4.3	Attributes for embedded content and images
10.4.4	Image maps
10.5	Bindings
10.5.1	Introduction
10.5.2	The <code>button</code> element

- 10.5.3 The `details` element
- 10.5.4 The `input` element as a text entry widget
- 10.5.5 The `input` element as domain-specific widgets
- 10.5.6 The `input` element as a range control
- 10.5.7 The `input` element as a color well
- 10.5.8 The `input` element as a checkbox and radio button widgets
- 10.5.9 The `input` element as a file upload control
- 10.5.10 The `input` element as a button
- 10.5.11 The `marquee` element
- 10.5.12 The `meter` element
- 10.5.13 The `progress` element
- 10.5.14 The `select` element
- 10.5.15 The `textarea` element
- 10.5.16 The `keygen` element
- 10.6 Frames and framesets
- 10.7 Interactive media
 - 10.7.1 Links, forms, and navigation
 - 10.7.2 The `title` attribute
 - 10.7.3 Editing hosts
 - 10.7.4 Text rendered in native user interfaces
- 10.8 Print media
- 10.9 Unstyled XML documents

11 Obsolete features

- 11.1 Obsolete but conforming features
 - 11.1.1 Warnings for obsolete but conforming features
- 11.2 Non-conforming features
- 11.3 Requirements for implementations
 - 11.3.1 The `applet` element
 - 11.3.2 The `marquee` element
 - 11.3.3 Frames
 - 11.3.4 Application caches
 - 11.3.4.1 Parsing cache manifests
 - 11.3.4.2 Downloading or updating an application cache
 - 11.3.4.3 The application cache selection algorithm
 - 11.3.4.4 Changes to the networking model
 - 11.3.4.5 Expiring application caches
 - 11.3.4.6 Disk space
 - 11.3.4.7 Security concerns with offline applications caches
 - 11.3.4.8 Application cache API
 - 11.3.5 Other elements, attributes and APIs

12 IANA considerations

- 12.1 `text/html`
- 12.2 `multipart/x-mixed-replace`
- 12.3 `application/xhtml+xml`
- 12.4 `web+` scheme prefix

Index

Terms defined by this specification

Terms defined by reference

Elements

Element content categories

Attributes

Element Interfaces

Events

IDL Index

References

Normative References

Informative References

Changes

Changes since HTML 5.1

Errata corrected

Changes since HTML 5.0

Features added

Features removed

Changes to existing features

Acknowledgements

§ 1. Introduction

§ 1.1. Background

This section is non-normative.

HTML is the World Wide Web’s core markup language. Originally, HTML was primarily designed as a language for semantically describing scientific documents. Its general design, however, has enabled it to be adapted, over the subsequent years, to describe a number of other types of documents and even applications.

§ 1.2. Audience

This section is non-normative.

This specification is intended for authors of documents and scripts that use the features defined in this specification, implementors of tools that operate on pages that use the features defined in this specification, and individuals wishing to establish the correctness of documents or implementations with respect to the requirements of this specification.

This document is probably not suited to readers who do not already have at least a passing familiarity with Web technologies, as in places it sacrifices clarity for precision, and brevity for completeness. More approachable tutorials and authoring guides can provide a gentler introduction to the topic.

In particular, familiarity with the basics of DOM is necessary for a complete understanding of some of the more technical parts of this specification. An understanding of Web IDL, HTTP, XML, Unicode, character encodings, JavaScript, and CSS will also be helpful in places but is not essential.

§ 1.3. Scope

This section is non-normative.

This specification is limited to providing a semantic-level markup language and associated semantic-level scripting APIs for authoring accessible pages on the Web ranging from static documents to dynamic applications.

The scope of this specification does not include providing mechanisms for media-specific customization of presentation (although default rendering rules for Web browsers are included at the end of this specification, and several mechanisms for hooking into CSS are provided as part of the language).

The scope of this specification is not to describe an entire operating system. In particular, hardware configuration software, image manipulation tools, and applications that users would be expected to use with high-end workstations on a daily basis are out of scope. In terms of applications, this specification is targeted specifically at applications that would be expected to be used by users on an occasional basis, or regularly but from disparate locations, with low CPU requirements. Examples of such applications include online purchasing systems, searching systems, games (especially multiplayer online games), pub-

lic telephone books or address books, communications software (e-mail clients, instant messaging clients, discussion software), document editing software, etc.

§ 1.4. History

This section is non-normative.

For its first five years (1990-1995), HTML went through a number of revisions and experienced a number of extensions, primarily hosted first at CERN, and then at the IETF.

With the creation of the W3C, HTML's development changed venue again. A first abortive attempt at extending HTML in 1995 known as HTML 3.0 then made way to a more pragmatic approach known as HTML 3.2, which was completed in 1997. HTML 4.01 quickly followed later that same year.

The following year, the W3C membership decided to stop evolving HTML and instead begin work on an XML-based equivalent, called XHTML. This effort started with a reformulation of HTML 4.01 in XML, known as XHTML 1.0, which added no new features except the new serialization, and which was completed in 2000. After XHTML 1.0, the W3C's focus turned to making it easier for other working groups to extend XHTML, under the banner of XHTML Modularization. In parallel with this, the W3C also worked on a new language that was not compatible with the earlier HTML and XHTML languages, calling it XHTML 2.0.

Around the time that HTML's evolution was stopped in 1998, parts of the API for HTML developed by browser vendors were specified and published under the name DOM Level 1 (in 1998) and DOM Level 2 Core and DOM Level 2 HTML (starting in 2000 and culminating in 2003). These efforts then petered out, with some DOM Level 3 specifications published in 2004 but the working group being closed before all the Level 3 drafts were completed.

In 2003, the publication of XForms, a technology which was positioned as the next generation of Web forms, sparked a renewed interest in evolving HTML itself, rather than finding replacements for it. This interest was borne from the realization that XML's deployment as a Web technology was limited to entirely new technologies (like RSS and later Atom), rather than as a replacement for existing deployed technologies (like HTML).

A proof of concept to show that it was possible to extend HTML 4.01's forms to provide many of the features that XForms 1.0 introduced, without requiring browsers to implement rendering engines that were incompatible with existing HTML Web pages, was the first result of this renewed interest. At this early stage, while the draft was already publicly available, and input was already being solicited from all sources, the specification was only under Opera Software's copyright.

The idea that HTML's evolution should be reopened was tested at a W3C workshop in 2004, where some of the principles that underlie the HTML work (described below), as well as the aforementioned early draft proposal covering just forms-related features, were presented to the W3C jointly by Mozilla and Opera. The proposal was rejected on the grounds that the proposal conflicted with the previously chosen direction for the Web's evolution; the W3C staff and membership voted to continue developing XML-based replacements instead.

Shortly thereafter, Apple, Mozilla, and Opera jointly announced their intent to continue working on the effort under the umbrella of a new venue called the WHATWG. A public mailing list was created, and the draft was moved to the WHATWG site. The copyright was subsequently amended to be jointly owned by all three vendors, and to allow reuse of the specification.

The WHATWG was based on several core principles, in particular that technologies need to be backwards compatible, that specifications and implementations need to match even if this means changing the specification rather than the implementations, and that specifications need to be detailed enough that implementations can achieve complete interoperability without reverse-engineering each other.

The latter requirement in particular required that the scope of the HTML specification include what had previously been specified in three separate documents: HTML 4.01, XHTML 1.1, and DOM Level 2 HTML. It also meant including significantly more detail than had previously been considered the norm.

In 2006, the W3C indicated an interest to participate in the development of HTML 5.0 after all, and in 2007 formed a working group chartered to work with the WHATWG on the development of the HTML specification. Apple, Mozilla, and Opera allowed the W3C to publish the specification under the W3C copyright, while keeping a version with the less restrictive license on the WHATWG site.

For a number of years, both groups then worked together under the same editor: Ian Hickson. In 2011, the groups came to the conclusion that they had different goals: the W3C wanted to draw a line in the sand for features for a HTML 5.0 Recommendation, while the WHATWG wanted to continue working on a Continually Updated Specification for HTML, continuously maintaining the specification and adding new features. In mid 2012, a new editing team was introduced at the W3C to take care of creating a HTML 5.0 Recommendation and prepare a Working Draft for the next HTML version.

Since then, the W3C HTML WG has been cherry picking patches from the WHATWG that resolved bugs registered on the W3C HTML specification or more accurately represented implemented reality in user agents. At time of publication of this document, patches from the [WHATWG HTML specification](#) have been merged until January 12, 2016. The W3C HTML editors have also added patches that resulted from discussions and decisions made by the W3C HTML WG as well as bug fixes from bugs not shared by the WHATWG.

A separate document is published to document the differences between the HTML specified in this document and the language described in the HTML 4.01 specification. [\[HTML5-DIFF\]](#)

§ 1.5. Design notes

This section is non-normative.

It must be admitted that many aspects of HTML appear at first glance to be nonsensical and inconsistent.

HTML, its supporting DOM APIs, as well as many of its supporting technologies, have been developed over a period of several decades by a wide array of people with different priorities who, in many cases, did not know of each other's existence.

Features have thus arisen from many sources, and have not always been designed in especially consistent ways. Furthermore, because of the unique characteristics of the Web, implementation bugs have often become de-facto, and now de-jure, standards, as content is often unintentionally written in ways that rely on them before they can be fixed.

Despite all this, efforts have been made to adhere to certain design goals. These are described in the next few subsections.

§ 1.5.1. Serializability of script execution

This section is non-normative.

To avoid exposing Web authors to the complexities of multithreading, the HTML and DOM APIs are designed such that no script can ever detect the simultaneous execution of other scripts. Even with [workers](#), the intent is that the behavior of implementations can be thought of as completely serializing the execution of all scripts in all [browsing contexts](#).

§ 1.5.2. Compliance with other specifications

This section is non-normative.

This specification interacts with and relies on a wide variety of other specifications. In certain circumstances, unfortunately, conflicting needs have led to this specification violating the requirements of these other specifications. Whenever this has occurred, the transgressions have each been noted as a "**willful violation**", and the reason for the violation has been noted.

§ 1.5.3. Extensibility

This section is non-normative.

HTML has a wide array of extensibility mechanisms that can be used for adding semantics in a safe manner:

- Authors can use the [class](#) attribute to extend elements, effectively creating their own elements, while using the most applicable existing "real" HTML element, so that browsers and other tools that don't know of the extension can still support it somewhat well. This is the tack used by microformats, for example.
- Authors can include data for inline client-side scripts or server-side site-wide scripts to process using the `data-*=` attributes. These are guaranteed to never be touched by browsers, and allow scripts to include data on HTML elements

that scripts can then look for and process.

- Authors can use the `<meta name="" content="">` mechanism to include page-wide metadata by registering [extensions to the predefined set of metadata names](#).
- Authors can use the `rel=""` mechanism to annotate links with specific meanings by registering [extensions to the predefined set of link types](#). This is also used by microformats.
- Authors can embed raw data using the `<script type="">` mechanism with a custom type, for further handling by inline or server-side scripts.
- Authors can create [plugins](#) and invoke them using the `<embed>` element. This is how Flash works.
- Authors can extend APIs using the JavaScript prototyping mechanism. This is widely used by script libraries, for instance.

§ 1.6. HTML vs XHTML

This section is non-normative.

This specification defines an abstract language for describing documents and applications, and some APIs for interacting with in-memory representations of resources that use this language.

The in-memory representation is known as "DOM HTML", or "the DOM" for short.

There are various concrete syntaxes that can be used to transmit resources that use this abstract language, two of which are defined in this specification.

The first such concrete syntax is the HTML syntax. This is the format suggested for most authors. It is compatible with most legacy Web browsers. If a document is transmitted with the [text/html MIME type](#), then it will be processed as an HTML document by Web browsers. This specification defines the latest version of the HTML syntax, known as "HTML 5.1".

The second concrete syntax is the XHTML syntax, which is an application of XML. When a document is transmitted with an [XML MIME type](#), such as [application/xhtml+xml](#), then it is treated as an XML document by Web browsers, to be parsed by an XML processor. Authors are reminded that the processing for XML and HTML differs; in particular, even minor syntax errors will prevent a document labeled as XML from being rendered fully, whereas they would be ignored in the HTML syntax. This specification defines the latest version of the XHTML syntax, known as "XHTML 5.1".

The DOM, the HTML syntax, and the XHTML syntax cannot all represent the same content. For example, namespaces cannot be represented using the HTML syntax, but they are supported in the DOM and in the XHTML syntax. Similarly, documents that use the `<noscript>` feature can be represented using the HTML syntax, but cannot be represented with the DOM or in the XHTML syntax. Comments that contain the string "-->" can only be represented in the DOM, not in the HTML and XHTML syntaxes.

§ 1.7. Structure of this specification

This section is non-normative.

This specification is divided into the following major sections:

[§1 Introduction](#)

Non-normative materials providing a context for the HTML standard.

[§2 Common infrastructure](#)

The conformance classes, algorithms, definitions, and the common underpinnings of the rest of the specification.

[§3 Semantics, structure, and APIs of HTML documents](#)

Documents are built from elements. These elements form a tree using the DOM. This section defines the features of this DOM, as well as introducing the features common to all elements, and the concepts used in defining elements.

[§4 The elements of HTML](#)

Each element has a predefined meaning, which is explained in this section. Rules for authors on how to use the ele-

ment, along with user agent requirements for how to handle each element, are also given. This includes large signature features of HTML such as video playback and subtitles, form controls and form submission, and a 2D graphics API known as the HTML canvas.

[§5 User interaction](#)

HTML documents can provide a number of mechanisms for users to interact with and modify content, which are described in this section, such as how focus works, and drag-and-drop.

[§6 Loading Web pages](#)

HTML documents do not exist in a vacuum — this section defines many of the features that affect environments that deal with multiple pages, such as Web browsers and offline caching of Web applications.

[§7 Web application APIs](#)

This section introduces basic features for scripting of applications in HTML.

[§8 The HTML syntax](#)

[§9 The XHTML syntax](#)

All of these features would be for naught if they couldn't be represented in a serialized form and sent to other people, and so these sections define the syntaxes of HTML and XHTML, along with rules for how to parse content using those syntaxes.

[§10 Rendering](#)

This section defines the default rendering rules for Web browsers.

There are also some appendices, listing [§11 Obsolete features](#) and [§12 IANA considerations](#), and several indices.

[§ 1.7.1. How to read this specification](#)

This specification should be read like all other specifications. First, it should be read cover-to-cover, multiple times. Then, it should be read backwards at least once. Then it should be read by picking random sections from the contents list and following all the cross-references.

As described in the conformance requirements section below, this specification describes conformance criteria for a variety of conformance classes. In particular, there are conformance requirements that apply to *producers*, for example authors and the documents they create, and there are conformance requirements that apply to *consumers*, for example Web browsers. They can be distinguished by what they are requiring: a requirement on a producer states what is allowed, while a requirement on a consumer states how software is to act.

EXAMPLE 1

For example, "the foo attribute's value must be a [valid integer](#)" is a requirement on producers, as it lays out the allowed values; in contrast, the requirement "the foo attribute's value must be parsed using the [rules for parsing integers](#)" is a requirement on consumers, as it describes how to process the content.

Requirements on producers have no bearing whatsoever on consumers.

EXAMPLE 2

Continuing the above example, a requirement stating that a particular attribute's value is constrained to being a [valid integer](#) emphatically does *not* imply anything about the requirements on consumers. It might be that the consumers are in fact required to treat the attribute as an opaque string, completely unaffected by whether the value conforms to the requirements or not. It might be (as in the previous example) that the consumers are required to parse the value using specific rules that define how invalid (non-numeric in this case) values are to be processed.

[§ 1.7.2. Typographic conventions](#)

This is a definition, requirement, or explanation.

NOTE:

This is a note.

EXAMPLE 3

This is an example.

This is an open issue.

⚠Warning! This is a warning.

```
interface Example {
    // this is an IDL definition
};
```

This definition is non-normative. Implementation requirements are given below this definition.

variable = ***object*** . ***method***([***optionalArgument***])

This is a note to authors describing the usage of an interface.

```
/* this is a CSS fragment */
,
```

The defining instance of a term is marked up like **this**. Uses of that term are marked up like [this](#) or like [this](#).

The defining instance of an element, attribute, or API is marked up like **this**. References to that element, attribute, or API are marked up like [<this>](#).

Other code fragments are marked up like `this`.

Byte sequences with bytes in the range 0x00 to 0x7F, inclusive, are marked up like ``this``.

Variables are marked up like *this*.

In an algorithm, steps in [synchronous sections](#) are marked with ⏸.

In some cases, requirements are given in the form of lists with conditions and corresponding requirements. In such cases, the requirements that apply to a condition are always the first set of requirements that follow the condition, even in the case of there being multiple sets of conditions for those requirements. Such cases are presented as follows:

↪ **This is a condition**

↪ **This is another condition**

This is the requirement that applies to the conditions above.

↪ **This is a third condition**

This is the requirement that applies to the third condition.

§ 1.8. Privacy concerns

This section is non-normative.

Some features of HTML trade user convenience for a measure of user privacy.

In general, due to the Internet's architecture, a user can be distinguished from another by the user's IP address. IP addresses do not perfectly match to a user; as a user moves from device to device, or from network to network, their IP address will change; similarly, NAT routing, proxy servers, and shared computers enable packets that appear to all come from a single IP address to actually map to multiple users. Technologies such as onion routing can be used to further anonymize requests so that requests from a single user at one node on the Internet appear to come from many disparate parts of the network.

However, the IP address used for a user's requests is not the only mechanism by which a user's requests could be related to each other.

Cookies, for example, are designed specifically to enable this, and are the basis of most of the Web's session features that enable you to log into a site with which you have an account.

Application caches have similar implications with respect to privacy, for example if the site can identify the user when providing the cache, it can store data in the cache that can be used for cookie resurrection.

There are other mechanisms that are more subtle. Certain characteristics of a user's system can be used to distinguish groups of users from each other; by collecting enough such information, an individual user's browser's "digital fingerprint" can be computed, which can be as good, if not better, as an IP address in ascertaining which requests are from the same user.

Grouping requests in this manner, especially across multiple sites, can be used for both benign (and even arguably positive) purposes, as well as for malevolent purposes. An example of a reasonably benign purpose would be determining whether a particular person seems to prefer sites with dog illustrations as opposed to sites with cat illustrations (based on how often they visit the sites in question) and then automatically using the preferred illustrations on subsequent visits to participating sites. Malevolent purposes, however, could include governments combining information such as the person's home address (determined from the addresses they use when getting driving directions on one site) with their apparent political affiliations (determined by examining the forum sites that they participate in) to determine whether the person should be prevented from voting in an election.

Since the malevolent purposes can be remarkably evil, user agent implementors are encouraged to consider how to provide their users with tools to minimize leaking information that could be used to fingerprint a user.

Unfortunately, as the first paragraph in this section implies, sometimes there is great benefit to be derived from exposing the very information that can also be used for fingerprinting purposes, so it's not as easy as simply blocking all possible leaks. For instance, the ability to log into a site to post under a specific identity requires that the user's requests be identifiable as all being from the same user. More subtly, though, information such as how wide text is, which is necessary for many effects that involve drawing text onto a canvas (e.g., any effect that involves drawing a border around the text) also leaks information that can be used to group a user's requests. (In this case, by potentially exposing, via a brute force search, which fonts a user has installed, information which can vary considerably from user to user.)

Features in this specification which can be **used to fingerprint the user** are marked as this paragraph is. 

Other features in the platform can be used for the same purpose, though, including, though not limited to:

- The exact list of which features a user agents supports.
- The maximum allowed stack depth for recursion in script.
- Features that describe the user's environment, like Media Queries and the [Screen](#) object. [\[MEDIAQ\]](#) [\[CSSOM-VIEW\]](#)
- The user's time zone.

§ 1.9. A quick introduction to HTML

This section is non-normative.

A basic HTML document looks like this:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sample page</title>
  </head>
  <body>
    <h1>Sample page</h1>
    <p>This is a <a href="demo.html">simple</a> sample.</p>
    <!-- this is a comment -->
  </body>
</html>
```

HTML documents consist of a tree of elements and text. Each element is denoted in the source by a [start tag](#), such as "`<<body>>`", and an [end tag](#), such as "`<</body>>`". (Certain start tags and end tags can in certain cases be [omitted](#) and are implied by other tags.)

Tags have to be nested such that elements are all completely within each other, without overlapping:

```
<p>This is <em>very <strong>wrong</em>!</strong></p>
```

```
<p>This <em>is <strong>correct</strong>.</em></p>
```

This specification defines a set of elements that can be used in HTML, along with rules about the ways in which the elements can be nested.

Elements can have attributes, which control how the elements work. In the example below, there is a [hyperlink](#), formed using the `<a>` element and its `href` attribute:

```
<a href="demo.html">simple</a>
```

[Attributes](#) are placed inside the start tag, and consist of a [name](#) and a [value](#), separated by an "=" character. The attribute value can remain [unquoted](#) if it doesn't contain [space characters](#) or any of " ' ` = < or >. Otherwise, it has to be quoted using either single or double quotes. The value, along with the "=" character, can be omitted altogether if the value is the empty string.

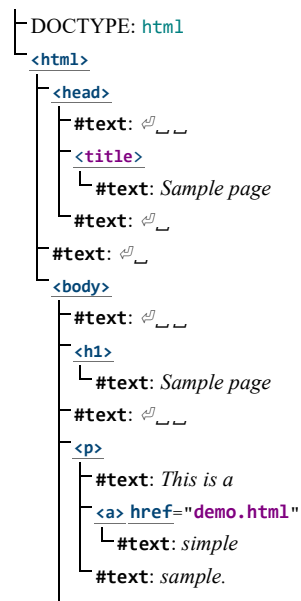
```
<!-- empty attributes -->
<input name=address disabled>
<input name=address disabled="">

<!-- attributes with a value -->
<input name=address maxlength=200>
<input name=address maxlength='200'>
<input name=address maxlength="200">
```

HTML user agents (e.g., Web browsers) then [parse](#) this markup, turning it into a DOM (Document Object Model) tree. A DOM tree is an in-memory representation of a document.

DOM trees contain several kinds of nodes, in particular a [DocumentType](#) node, [Element](#) nodes, [Text](#) nodes, [Comment](#) nodes, and in some cases [ProcessingInstruction](#) nodes.

The [markup snippet at the top of this section](#) would be turned into the following DOM tree:



```

#text: ␣␣␣
#comment: this is a comment
#text: ␣␣␣

```

The [root element](#) of this tree is the [<html>](#) element, which is the element always found at the root of HTML documents. It contains two elements, [<head>](#) and [<body>](#), as well as a [Text](#) node between them.

There are many more [Text](#) nodes in the DOM tree than one would initially expect, because the source contains a number of spaces (represented here by "␣") and line breaks ("␣") that all end up as [Text](#) nodes in the DOM. However, for historical reasons not all of the spaces and line breaks in the original markup appear in the DOM. In particular, all the whitespace before [<head>](#) start tag ends up being dropped silently, and all the whitespace after the [<body>](#) end tag ends up placed at the end of the [<body>](#).

The [<head>](#) element contains a [<title>](#) element, which itself contains a [Text](#) node with the text "Sample page". Similarly, the [<body>](#) element contains an [<h1>](#) element, a [<p>](#) element, and a comment.



This DOM tree can be manipulated from scripts in the page. Scripts (typically in JavaScript) are small programs that can be embedded using the [<script>](#) element or using [event handler content attributes](#). For example, here is a form with a script that sets the value of the form's [<output>](#) element to say "Hello World"

```

<form name="main">
  Result: <output name="result"></output>
  <script>
    document.forms.main.elements.result.value = 'Hello World';
  </script>
</form>

```

Each element in the DOM tree is represented by an object, and these objects have APIs so that they can be manipulated. For instance, a link (e.g., the [<a>](#) element in the tree above) can have its ["href"](#) attribute changed in several ways:

```

var a = document.links[0]; // obtain the first link in the document
a.href = 'sample.html'; // change the destination URL of the link
a.protocol = 'https'; // change just the scheme part of the URL
a.setAttribute('href', 'https://example.com/'); // change the content attribute directly

```

Since DOM trees are used as the way to represent HTML documents when they are processed and presented by implementations (especially interactive implementations like Web browsers), this specification is mostly phrased in terms of DOM trees, instead of the markup described above.



HTML documents represent a media-independent description of interactive content. HTML documents might be rendered to a screen, or through a speech synthesizer, or on a braille display. To influence exactly how such rendering takes place, authors can use a styling language such as CSS.

In the following example, the page has been made yellow-on-blue using CSS.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sample styled page</title>
    <style>
      body { background: navy; color: yellow; }
    </style>
  </head>
  <body>
    <h1>Sample styled page</h1>
    <p>This page is just a demo.</p>
  </body>
</html>
```

For more details on how to use HTML, authors are encouraged to consult tutorials and guides. Some of the examples included in this specification might also be of use, but the novice author is cautioned that this specification, by necessity, defines the language with a level of detail that might be difficult to understand at first.

§ 1.9.1. Writing secure applications with HTML

This section is non-normative.

When HTML is used to create interactive sites, care needs to be taken to avoid introducing vulnerabilities through which attackers can compromise the integrity of the site itself or of the site's users.

A comprehensive study of this matter is beyond the scope of this document, and authors are strongly encouraged to study the matter in more detail. However, this section attempts to provide a quick introduction to some common pitfalls in HTML application development.

The security model of the Web is based on the concept of "origins", and correspondingly many of the potential attacks on the Web involve cross-origin actions. [\[ORIGIN\]](#)

Not validating user input

Cross-site scripting (XSS)

SQL injection

When accepting untrusted input, e.g., user-generated content such as text comments, values in URL parameters, messages from third-party sites, etc, it is imperative that the data be validated before use, and properly escaped when displayed. Failing to do this can allow a hostile user to perform a variety of attacks, ranging from the potentially benign, such as providing bogus user information like a negative age, to the serious, such as running scripts every time a user looks at a page that includes the information, potentially propagating the attack in the process, to the catastrophic, such as deleting all data in the server.

When writing filters to validate user input, it is imperative that filters always be safelist-based, allowing known-safe constructs and disallowing all other input. Blocklist-based filters that disallow known-bad inputs and allow everything else are not secure, as not everything that is bad is yet known (for example, because it might be invented in the future).

EXAMPLE 4

For example, suppose a page looked at its URL's query string to determine what to display, and the site then redirected the user to that page to display a message, as in:

```
<ul>
  <li><a href="message.cgi?say=Hello">Say Hello</a>
  <li><a href="message.cgi?say=Welcome">Say Welcome</a>
  <li><a href="message.cgi?say=Kittens">Say Kittens</a>
</ul>
```

If the message was just displayed to the user without escaping, a hostile attacker could then craft a URL that contained a script element:

```
https://example.com/message.cgi?say=%3Cscript%3Ealert%28%270h%20no%21%27%29%3C/script%3E
```

If the attacker then convinced a victim user to visit this page, a script of the attacker's choosing would run on the page. Such a script could do any number of hostile actions, limited only by what the site offers: if the site is an e-commerce shop, for instance, such a script could cause the user to unknowingly make arbitrarily many unwanted purchases.

This is called a cross-site scripting attack.

There are many constructs that can be used to try to trick a site into executing code. Here are some that authors are encouraged to consider when writing safelist filters:

- When allowing harmless-seeming elements like `<img>`, exercise the principle of least-privilege and limit the element's attributes to only those that are needed (e.g., via a safelist). If one allowed all attributes then an attacker could, for instance, use the `onload` attribute to run arbitrary script.
- When allowing URLs to be provided (e.g., for links), the scheme of each URL also needs to be explicitly safelisted, as there are many schemes that can be abused. The most prominent example is "javascript:", but user agents can implement (and indeed, have historically implemented) others.
- Allowing a `<base>` element to be inserted means any `<script>` elements in the page with relative links can be hijacked, and similarly that any form submissions can get redirected to a hostile site.

Cross-site request forgery (CSRF)

If a site allows a user to make form submissions with user-specific side-effects, for example posting messages on a forum under the user's name, making purchases, or applying for a passport, it is important to verify that the request was made by the user intentionally, rather than by another site tricking the user into making the request unknowingly.

This problem exists because HTML forms can be submitted to other origins.

Sites can prevent such attacks by populating forms with user-specific hidden tokens, or by checking Origin headers on all requests.

Clickjacking

A page that provides users with an interface to perform actions that the user might not wish to perform needs to be designed so as to avoid the possibility that users can be tricked into activating the interface.

One way that a user could be so tricked is if a hostile site places the victim site in a small `<iframe>` and then convinces the user to click, for instance by having the user play a reaction game. Once the user is playing the game, the hostile site can quickly position the `<iframe>` under the mouse cursor just as the user is about to click, thus tricking the user into clicking the victim site's interface.

To avoid this, sites that do not expect to be used in frames are encouraged to only enable their interface if they detect that they are not in a frame (e.g., by comparing the `Window` object to the value of the `top` attribute).

§ **1.9.2. Common pitfalls to avoid when using the scripting APIs**

This section is non-normative.

Scripts in HTML have "run-to-completion" semantics, meaning that the browser will generally run the script uninterrupted before doing anything else, such as firing further events or continuing to parse the document.

On the other hand, parsing of HTML files happens [in parallel](#) and incrementally, meaning that the parser can pause at any point to let scripts run. This is generally a good thing, but it does mean that authors need to be careful to avoid hooking event handlers after the events could have possibly fired.

There are two techniques for doing this reliably: use [event handler content attributes](#), or create the element and add the event handlers in the same script. The latter is safe because, as mentioned earlier, scripts are run to completion before further events can fire.

EXAMPLE 5

One way this could manifest itself is with [](#) elements and the `load` event. The event could fire as soon as the element has been parsed, especially if the image has already been cached (which is common).

Here, the author uses the [onload](#) handler on an [](#) element to catch the [load](#) event:

```

```

If the element is being added by script, then so long as the event handlers are added in the same script, the event will still not be missed:

```
<script>
var img = new Image();
img.src = 'games.png';
img.alt = 'Games';
img.onload = gamesLogoHasLoaded;
// img.addEventListener('load', gamesLogoHasLoaded, false); // would work also
</script>
```

However, if the author first created the [](#) element and then in a separate script added the event listeners, there's a chance that the `load` event would be fired in between, leading it to be missed:

```
<!-- Do not use this style, it has a race condition! -->

<!-- the 'load' event might fire here while the parser is taking a
      break, in which case you will not see it! -->
<script>
var img = document.getElementById('games');
img.onload = gamesLogoHasLoaded; // might never fire!
</script>
```

§ 1.9.3. How to catch mistakes when writing HTML: validators and conformance checkers

This section is non-normative.

Authors are encouraged to make use of conformance checkers (also known as *validators*) to catch common mistakes. The W3C provides a number of online validation services, including the [Nu Markup Validation Service](#).

§ 1.10. Conformance requirements for authors

This section is non-normative.

Unlike previous versions of the HTML specification, this specification defines in some detail the required processing for invalid documents as well as valid documents.

However, even though the processing of invalid content is in most cases well-defined, conformance requirements for documents are still important: in practice, interoperability (the situation in which all implementations process particular content

in a reliable and identical or equivalent way) is not the only goal of document conformance requirements. This section details some of the more common reasons for still distinguishing between a [conforming document](#) and one with errors.

§ 1.10.1. Presentational markup

This section is non-normative.

The majority of presentational features from previous versions of HTML are no longer allowed. Presentational markup in general has been found to have a number of problems:

The use of presentational elements leads to poorer accessibility

While it is possible to use presentational markup in a way that provides users of assistive technologies (ATs) with an acceptable experience (e.g., using ARIA), doing so is significantly more difficult than doing so when using semantically-appropriate markup. Furthermore, presentational markup does not guarantee accessibility for users of non-AT, non-graphical user agents (such as text-mode browsers).

Using media-independent markup, on the other hand, provides an easy way for documents to be authored in such a way that they are "accessible" for more users (e.g., users of text browsers).

Higher cost of maintenance

It is significantly easier to maintain a site written in such a way that the markup is style-independent. For example, changing the color of a site that uses `<font color="">` throughout requires changes across the entire site, whereas a similar change to a site based on CSS can be done by changing a single file.

Larger document sizes

Presentational markup tends to be much more redundant, and thus results in larger document sizes.

For those reasons, presentational markup has been removed from HTML in this version. This change should not come as a surprise; HTML 4.0 deprecated presentational markup many years ago and provided a mode (HTML Transitional) to help authors move away from presentational markup; later, XHTML 1.1 went further and obsoleted those features altogether.

The only remaining presentational markup features in HTML are the [style](#) attribute and the `<style>` element. Use of the [style](#) attribute is somewhat discouraged in production environments, but it can be useful for rapid prototyping (where its rules can be directly moved into a separate style sheet later) and for providing specific styles in unusual cases where a separate style sheet would be inconvenient. Similarly, the `<style>` element can be useful for grouping or for page-specific styles, but in general an external style sheet is likely to be more convenient when the styles apply to multiple pages.

It is also worth noting that some elements that were previously presentational have been redefined in this specification to be media-independent: [](#), [<i>](#), [<hr>](#), [<s>](#), [<small>](#), and [<u>](#).

§ 1.10.2. Syntax errors

This section is non-normative.

The syntax of HTML is constrained to avoid a wide variety of problems.

Unintuitive error-handling behavior

Certain invalid syntax constructs, when parsed, result in DOM trees that are highly unintuitive.

EXAMPLE 6

For example, the following markup fragment results in a DOM with an `<hr>` element that is an *earlier* sibling of the corresponding `<table>` element:

```
<table><hr>...
```

Errors with optional error recovery

To allow user agents to be used in controlled environments without having to implement the more bizarre and convoluted error handling rules, user agents are permitted to fail whenever encountering a [parse error](#).

Errors where the error-handling behavior is not compatible with streaming user agents

Some error-handling behavior, such as the behavior for the `<table><hr>...` example mentioned above, are incompatible with streaming user agents (user agents that process HTML files in one pass, without storing state). To avoid interoperability problems with such user agents, any syntax resulting in such behavior is considered invalid.

Errors that can result in infoset coercion

When a user agent based on XML is connected to an HTML parser, it is possible that certain invariants that XML enforces, such as comments never containing two consecutive hyphens, will be violated by an HTML file. Handling this can require that the parser coerce the HTML DOM into an XML-compatible infoset. Most syntax constructs that require such handling are considered invalid.

Errors that result in disproportionately poor performance

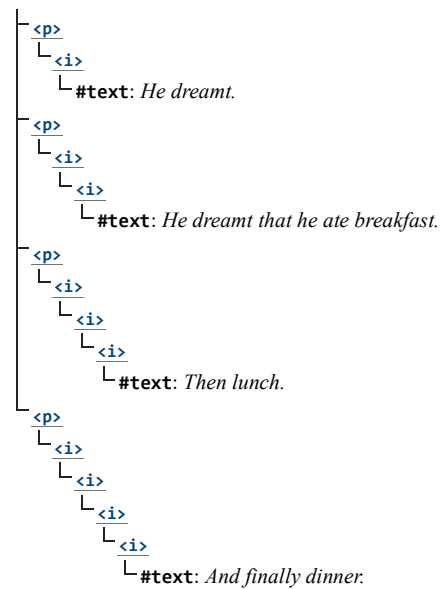
Certain syntax constructs can result in disproportionately poor performance. To discourage the use of such constructs, they are typically made non-conforming.

EXAMPLE 7

For example, the following markup results in poor performance, since all the unclosed `<i>` elements have to be re-constructed in each paragraph, resulting in progressively more elements in each paragraph:

```
<p><i>He dreamt.
<p><i>He dreamt that he ate breakfast.
<p><i>Then lunch.
<p><i>And finally dinner.
```

The resulting DOM for this fragment would be:



Errors involving fragile syntax constructs

There are syntax constructs that, for historical reasons, are relatively fragile. To help reduce the number of users who accidentally run into such problems, they are made non-conforming.

EXAMPLE 8

For example, the parsing of certain named character references in attributes happens even with the closing semicolon being omitted. It is safe to include an ampersand followed by letters that do not form a named character reference, but if the letters are changed to a string that *does* form a named character reference, they will be interpreted as that character instead.

In this fragment, the attribute's value is "?bill&ted":

```
<a href="?bill&ted">Bill and Ted</a>
```

In the following fragment, however, the attribute's value is actually "?art@", *not* the intended "?art&copy", because even without the final semicolon, "&copy" is handled the same as "©" and thus gets interpreted as "@":

```
<a href="?art&copy">Art and Copy</a>
```

To avoid this problem, all named character references are required to end with a semicolon, and uses of named character references without a semicolon are flagged as errors.

Thus, the correct way to express the above cases is as follows:

```
<a href="?bill&ted">Bill and Ted</a> <!-- &ted is ok, since it's not a named character reference -->
```

```
<a href="?art&amp;copy">Art and Copy</a> <!-- the & has to be escaped, since &copy is a named character reference -->
```

Errors involving known interoperability problems in legacy user agents

Certain syntax constructs are known to cause especially subtle or serious problems in legacy user agents, and are therefore marked as non-conforming to help authors avoid them.

EXAMPLE 9

For example, this is why the U+0060 GRAVE ACCENT character (`) is not allowed in unquoted attributes. In certain legacy user agents, it is sometimes treated as a quote character.

EXAMPLE 10

Another example of this is the DOCTYPE, which is required to trigger [no-quirks mode](#), because the behavior of legacy user agents in [quirks mode](#) is often largely undocumented.

Errors that risk exposing authors to security attacks

Certain restrictions exist purely to avoid known security problems.

EXAMPLE 11

For example, the restriction on using UTF-7 exists purely to avoid authors falling prey to a known cross-site-scripting attack using UTF-7. [\[RFC2152\]](#)

Cases where the author's intent is unclear

Markup where the author's intent is very unclear is often made non-conforming. Correcting these errors early makes later maintenance easier.

EXAMPLE 12

For example, it is unclear whether the author intended the following to be an [<h1>](#) heading or an [<h2>](#) heading:

```
<h2>Contact details</h1>
```

Cases that are likely to be typos

When a user makes a simple typo, it is helpful if the error can be caught early, as this can save the author a lot of de-

bugging time. This specification therefore usually considers it an error to use element names, attribute names, and so forth, that do not match the names defined in this specification.

EXAMPLE 13

For example, if the author typed `<capton>` instead of `<caption>`, this would be flagged as an error and the author could correct the typo immediately.

Errors that could interfere with new syntax in the future

In order to allow the language syntax to be extended in the future, certain otherwise harmless features are disallowed.

EXAMPLE 14

For example, attributes in end tags are currently invalid and ignored. A future change to the language may make use of this syntax feature and can do so without conflicting with already-deployed (and valid!) content.

Some authors find it helpful to be in the practice of always quoting all attributes and always including all optional tags, preferring the consistency derived from such custom over the minor benefits of terseness afforded by making use of the flexibility of the HTML syntax. To aid such authors, conformance checkers can provide modes of operation wherein such conventions are enforced.

§ 1.10.3. Restrictions on content models and on attribute values

This section is non-normative.

Beyond the syntax of the language, this specification also places restrictions on how elements and attributes can be specified. These restrictions are present for similar reasons:

Errors involving content with dubious semantics

To avoid misuse of elements with defined meanings, content models are defined that restrict how elements can be nested when such nestings would be of dubious value.

EXAMPLE 15

For example, this specification disallows nesting a `<section>` element inside a `<kbd>` element, since it is highly unlikely for an author to indicate that an entire section should be keyed in.

Errors that involve a conflict in expressed semantics

Similarly, to draw the author's attention to mistakes in the use of elements, clear contradictions in the semantics expressed are also considered conformance errors.

EXAMPLE 16

In the fragments below, for example, the semantics are nonsensical: a separator cannot simultaneously be a cell, nor can a radio button be a progress bar.

```
<hr role="cell">
```

```
<input type=radio role=progressbar>
```

EXAMPLE 17

Another example is the restrictions on the content models of the `<ul>` element, which only allows `<li>` element children. Lists by definition consist just of zero or more list items, so if a `<ul>` element contains something other than an `<li>` element, it's not clear what was meant.

Cases where the default styles are likely to lead to confusion

Certain elements have default styles or behaviors that make certain combinations likely to lead to confusion. Where these have equivalent alternatives without this problem, the confusing combinations are disallowed.

EXAMPLE 18

For example, `<div>` elements are rendered as block boxes, and `<span>` elements as inline boxes. Putting a block box in an inline box is unnecessarily confusing; since either nesting just `<div>` elements, or nesting just `<span>` elements, or nesting `<span>` elements inside `<div>` elements all serve the same purpose as nesting a `<div>` element in a `<span>` element, but only the latter involves a block box in an inline box, the latter combination is disallowed.

EXAMPLE 19

Another example would be the way [interactive content](#) cannot be nested. For example, a `<button>` element cannot contain a `<textarea>` element. This is because the default behavior of such nesting interactive elements would be highly confusing to users. Instead of nesting these elements, they can be placed side by side.

Errors that indicate a likely misunderstanding of the specification

Sometimes, something is disallowed because allowing it would likely cause author confusion.

EXAMPLE 20

For example, setting the `disabled` attribute to the value "false" is disallowed, because despite the appearance of meaning that the element is enabled, it in fact means that the element is *disabled* (what matters for implementations is the presence of the attribute, not its value).

Errors involving limits that have been imposed merely to simplify the language

Some conformance errors simplify the language that authors need to learn.

EXAMPLE 21

For example, the `<area>` element's `shape` attribute, despite accepting both `circ` and `circle` values in practice as synonyms, disallows the use of the `circ` value, so as to simplify tutorials and other learning aids. There would be no benefit to allowing both, but it would cause extra confusion when teaching the language.

Errors that involve peculiarities of the parser

Certain elements are parsed in somewhat eccentric ways (typically for historical reasons), and their content model restrictions are intended to avoid exposing the author to these issues.

EXAMPLE 22

For example, a `<form>` element isn't allowed inside [phrasing content](#), because when parsed as HTML, a `<form>` element's start tag will imply a `<p>` element's end tag. Thus, the following markup results in two [paragraphs](#), not one:

```
<p>Welcome. <form><label>Name:</label> <input></form>
```

It is parsed exactly like the following:

```
<p>Welcome. </p><form><label>Name:</label> <input></form>
```

Errors that would likely result in scripts failing in hard-to-debug ways

Some errors are intended to help prevent script problems that would be hard to debug.

EXAMPLE 23

This is why, for instance, it is non-conforming to have two `id` attributes with the same value. Duplicate IDs lead to the wrong element being selected, with sometimes disastrous effects whose cause is hard to determine.

Errors that waste authoring time

Some constructs are disallowed because historically they have been the cause of a lot of wasted authoring time, and by encouraging authors to avoid making them, authors can save time in future efforts.

EXAMPLE 24

For example, a `<script>` element's `src` attribute causes the element's contents to be ignored. However, this isn't obvious, especially if the element's contents appear to be executable script — which can lead to authors spending a lot of time trying to debug the inline script without realizing that it is not executing. To reduce this problem, this specification makes it non-conforming to have executable script in a `<script>` element when the `src` attribute is present. This means that authors who are validating their documents are less likely to waste time with this kind of mistake.

Errors that involve areas that affect authors migrating to and from XHTML

Some authors like to write files that can be interpreted as both XML and HTML with similar results. Though this practice is discouraged in general due to the myriad of subtle complications involved (especially when involving scripting, styling, or any kind of automated serialization), this specification has a few restrictions intended to at least somewhat mitigate the difficulties. This makes it easier for authors to use this as a transitional step when migrating between HTML and XHTML.

EXAMPLE 25

For example, there are somewhat complicated rules surrounding the `lang` and `xml:lang` attributes intended to keep the two synchronized.

EXAMPLE 26

Another example would be the restrictions on the values of `xmlns` attributes in the HTML serialization, which are intended to ensure that elements in conforming documents end up in the same namespaces whether processed as HTML or XML.

Errors that involve areas reserved for future expansion

As with the restrictions on the syntax intended to allow for new syntax in future revisions of the language, some restrictions on the content models of elements and values of attributes are intended to allow for future expansion of the HTML vocabulary.

EXAMPLE 27

For example, limiting the values of the `target` attribute that start with an U+005F LOW LINE character (`_`) to only specific predefined values allows new predefined values to be introduced at a future time without conflicting with author-defined values.

Errors that indicate a mis-use of other specifications

Certain restrictions are intended to support the restrictions made by other specifications.

EXAMPLE 28

For example, requiring that attributes that take media query lists use only *valid* media query lists reinforces the importance of following the conformance rules of that specification.

§ 1.11. Suggested reading

This section is non-normative.

The following documents might be of interest to readers of this specification.

Character Model for the World Wide Web 1.0: Fundamentals [\[CHARMOD\]](#)

This Architectural Specification provides authors of specifications, software developers, and content developers with a common reference for interoperable text manipulation on the World Wide Web, building on the Universal Character Set, defined jointly by the Unicode Standard and ISO/IEC 10646. Topics addressed include use of the terms "character", "encoding" and "string", a reference processing model, choice and identification of character encodings, character escaping, and string indexing.

Unicode Security Considerations [\[UNICODE-SECURITY\]](#)

Because Unicode contains such a large number of characters and incorporates the varied writing systems of the world, incorrect usage can expose programs or systems to possible security attacks. This is especially important as more and more products are internationalized. This document describes some of the security considerations that programmers, system analysts, standards developers, and users should take into account, and provides specific recommendations to reduce the risk of problems.

Web Content Accessibility Guidelines (WCAG) 2.0 [\[WCAG20\]](#)

Web Content Accessibility Guidelines (WCAG) 2.0 covers a wide range of recommendations for making Web content more accessible. Following these guidelines will make content accessible to a wider range of people with disabilities, including blindness and low vision, deafness and hearing loss, learning disabilities, cognitive limitations, limited movement, speech disabilities, photosensitivity and combinations of these. Following these guidelines will also often make your Web content more usable to users in general.

Authoring Tool Accessibility Guidelines (ATAG) 2.0 [\[ATAG20\]](#)

This specification provides guidelines for designing Web content authoring tools that are more accessible for people with disabilities. An authoring tool that conforms to these guidelines will promote accessibility by providing an accessible user interface to authors with disabilities as well as by enabling, supporting, and promoting the production of accessible Web content by all authors.

User Agent Accessibility Guidelines (UAAG) 2.0 [\[UAAG20\]](#)

This document provides guidelines for designing user agents that lower barriers to Web accessibility for people with disabilities. User agents include browsers and other types of software that retrieve and render Web content. A user agent that conforms to these guidelines will promote accessibility through its own user interface and through other internal facilities, including its ability to communicate with other technologies (especially assistive technologies). Furthermore, all users, not just users with disabilities, should find conforming user agents to be more usable.

Polyglot Markup: HTML-Compatible XHTML Documents [\[HTML-POLYGLOT\]](#)

A document that uses polyglot markup is a document that is a stream of bytes that parses into identical document trees (with the exception of the xmlns attribute on the root element) when processed as HTML and when processed as XML. Polyglot markup that meets a well defined set of constraints is interpreted as compatible, regardless of whether they are processed as HTML or as XHTML, per the HTML specification. Polyglot markup uses a specific DOCTYPE, namespace declarations, and a specific case — normally lower case but occasionally camel case — for element and attribute names. Polyglot markup uses lower case for certain attribute values. Further constraints include those on empty elements, named entity references, and the use of scripts and style.

HTML Accessibility APIs Mappings 1.0 [\[HTML-AAM-1.0\]](#)

Defines how user agents map HTML 5.1 elements and attributes to platform accessibility APIs. Documenting these mappings promotes interoperable exposure of roles, states, properties, and events implemented by accessibility APIs and helps to ensure that this information appears in a manner consistent with author intent.

§ 2. Common infrastructure

§ 2.1. Terminology

This specification refers to both HTML and XML attributes and IDL attributes, often in the same context. When it is not clear which is being referred to, they are referred to as **content attributes** for HTML and XML attributes, and **IDL attributes** for those defined on IDL interfaces. Similarly, the term "properties" is used for both JavaScript object properties and CSS properties. When these are ambiguous they are qualified as **object properties** and **CSS properties** respectively.

Generally, when the specification states that a feature applies to [the HTML syntax](#) or [the XHTML syntax](#), it also includes the other. When a feature specifically only applies to one of the two languages, it is called out by explicitly stating that it does not apply to the other format, as in "for HTML, ... (this does not apply to XHTML)".

This specification uses the term **document** to refer to any use of HTML, ranging from short static documents to long essays or reports with rich multimedia, as well as to fully-fledged interactive applications. The term is used to refer both to

[Document](#) objects and their descendant DOM trees, and to serialized byte streams using the [HTML syntax](#) or [XHTML syntax](#), depending on context.

In the context of the DOM structures, the terms **HTML document** and [XML document](#) are used as defined in the DOM specification, and refer specifically to two different modes that [Document](#) objects can find themselves in. [\[DOM\]](#) (Such uses are always hyperlinked to their definition.)

In the context of byte streams, the term [HTML document](#) refers to resources labeled as [text/html](#), and the term **XML document** refers to resources labeled with an [XML MIME type](#).

The term **XHTML document** is used to refer to both [documents](#) in the [XML document](#) mode that contain element nodes in the [HTML namespace](#), and byte streams labeled with an [XML MIME type](#) that contain elements from the [HTML namespace](#), depending on context.



For simplicity, terms such as **shown**, **displayed**, and **visible** are used (sometimes) when referring to the way a document is rendered to the user. These terms are not meant to imply a visual medium; they must be considered to apply to other media in equivalent ways.

When an algorithm B says to return to another algorithm A, it implies that A called B. Upon returning to A, the implementation must continue from where it left off in calling B. Some algorithms run **in parallel**; this means that the algorithm's subsequent steps are to be run, one after another, at the same time as other logic in the specification (e.g., at the same time as the [event loop](#)). This specification does not define the precise mechanism by which this is achieved, be it time-sharing cooperative multitasking, fibers, threads, processes, using different hyperthreads, cores, CPUs, machines, etc. By contrast, an operation that is to run **immediately** must interrupt the currently running task, run itself, and then resume the previously running task.

The term "transparent black" refers to the color with red, green, blue, and alpha channels all set to zero.

§ 2.1.1. Resources

The specification uses the term **supported** when referring to whether a user agent has an implementation capable of decoding the semantics of an external resource. A format or type is said to be *supported* if the implementation can process an external resource of that format or type without critical aspects of the resource being ignored. Whether a specific resource is *supported* can depend on what features of the resource's format are in use.

EXAMPLE 29

For example, a PNG image would be considered to be in a supported format if its pixel data could be decoded and rendered, even if, unbeknownst to the implementation, the image also contained animation data.

EXAMPLE 30

An MPEG-4 video file would not be considered to be in a supported format if the compression format used was not supported, even if the implementation could determine the dimensions of the movie from the file's metadata.

What some specifications, in particular the HTTP specification, refer to as a *representation* is referred to in this specification as a **resource**. [\[HTTP\]](#)

The term **MIME type** is used to refer to what is sometimes called an *Internet media type* in protocol literature. The term **media type** in this specification is used to refer to the type of media intended for presentation, as used by the CSS specifications. [\[RFC2046\]](#) [\[MEDIAQ\]](#)

A string is a **valid MIME type** if it matches the [media-type](#) rule. In particular, a [valid mime type](#) may include MIME type parameters. [\[HTTP\]](#)

A string is a **valid MIME type with no parameters** if it matches the [media-type](#) rule, but does not contain any U+003B SEMICOLON characters (;). In other words, if it consists only of a type and subtype, with no MIME Type parameters. [\[HTTP\]](#)

The term **HTML MIME type** is used to refer to the [MIME type text/html](#).

A resource's **critical subresources** are those that the resource needs to have available to be correctly processed. Which resources are considered critical or not is defined by the specification that defines the resource's format.

The term [data: URL](#) refers to [URLs](#) that use the [data:](#) scheme. [\[RFC2397\]](#)

§ 2.1.2. XML

To ease migration from HTML to XHTML, user agents conforming to this specification will place elements in HTML in the <http://www.w3.org/1999/xhtml> namespace, at least for the purposes of the DOM and CSS. The term "**HTML elements**", when used in this specification, refers to any element in that namespace, and thus refers to both HTML and XHTML elements.

Except where otherwise stated, all elements defined or mentioned in this specification are in the [HTML namespace](#) ("<http://www.w3.org/1999/xhtml>"), and all attributes defined or mentioned in this specification have no namespace.

The term **element type** is used to refer to the set of elements that have a given local name and namespace. For example, [<button>](#) elements are elements with the element type [<button>](#), meaning they have the local name "[<button>](#)" and (implicitly as defined above) the [HTML namespace](#).

Attribute names are said to be **XML-compatible** if they match the [Name](#) production defined in XML and they contain no U+003A COLON characters (:). [\[XML\]](#)

The term **XML MIME type** is used to refer to the [MIME types](#) [text/xml](#), [application/xml](#), and any [MIME type](#) whose subtype ends with the four characters "+xml". [\[RFC7303\]](#)

§ 2.1.3. DOM trees

The **root element of a Document object** is that [Document](#)'s first element child, if any. If it does not have one then the [Document](#) has no root element.

The term **root element**, when not referring to a [Document](#) object's root element, means the furthest ancestor element node of whatever node is being discussed, or the node itself if it has no ancestors. When the node is a part of the document, then the node's [root element](#) is indeed the document's root element; however, if the node is not currently part of the document tree, the root element will be an orphaned node.

When an element's [root element](#) is the [root element of a Document object](#), it is said to be **in a Document**. An element is said to have been **inserted into a document** when its [root element](#) changes and is now the document's [root element](#). Analogously, an element is said to have been **removed from a document** when its [root element](#) changes from being the document's [root element](#) to being another element.

A node's **home subtree** is the subtree rooted at that node's [root element](#). When a node is [in a Document](#), its [home subtree](#) is that [Document](#)'s tree.

The [Document](#) of a [Node](#) (such as an element) is the [Document](#) that the [Node](#)'s [ownerDocument](#) IDL attribute returns. When a [Node](#) is [in a Document](#) then that [Document](#) is always the [Node](#)'s [Document](#), and the [Node](#)'s [ownerDocument](#) IDL attribute thus always returns that [Document](#).

The [Document](#) of a content attribute is the [Document](#) of the attribute's element.

The term **tree order** means a pre-order, depth-first traversal of DOM nodes involved (through the [parentNode/childNodes](#) relationship).

When it is stated that some element or attribute is **ignored**, or treated as some other value, or handled as if it was something else, this refers only to the processing of the node after it is in the DOM. A user agent must not mutate the DOM in such situations.

A content attribute is said to **change** value only if its new value is different than its previous value; setting an attribute to a value it already has does not change it.

When an attribute value, [Text](#) node, or string is described as **empty**, it means that the length of the text is zero (i.e., not

even containing spaces or [control characters](#)).

An element's **child text content** is the concatenation of the [data](#) of all the [Text](#) nodes that are children of the element (ignoring any other nodes such as comments or elements), in [tree order](#).

A [node *A* is inserted](#) into a node *B* when the [insertion steps](#) are invoked with *A* as the argument and *A*'s new parent is *B*. Similarly, a **node *A* is removed** from a node *B* when the [removing steps](#) are invoked with *A* as the *removedNode* argument and *B* as the *oldParent* argument.

§ 2.1.4. Scripting

The construction "a Foo object", where Foo is actually an interface, is sometimes used instead of the more accurate "an object implementing the interface Foo".

An IDL attribute is said to be **getting** when its value is being retrieved (e.g., by author script), and is said to be **setting** when a new value is assigned to it.

If a DOM object is said to be **live**, then the attributes and methods on that object must operate on the actual underlying data, not a snapshot of the data.

In the contexts of events, the terms *fire* and *dispatch* are used as defined in the DOM specification: **firing** an event means to create and [dispatch](#) it, and **dispatching** an event means to follow the steps that propagate the event through the tree. The term **trusted event** is used to refer to events whose [isTrusted](#) attribute is initialized to true. [\[DOM\]](#)

§ 2.1.5. Plugin Content Handlers

The term **plugin** refers to a user-agent defined set of content handlers that can be used by the user agent. The content handlers can take part in the user agent's rendering of a [Document](#) object, but that neither act as [child browsing contexts](#) of the [Document](#) nor introduce any [Node](#) objects to the [Document](#)'s DOM.

Typically such content handlers are provided by third parties, though a user agent can also designate built-in content handlers as [plugins](#).

A user agent must not consider the types `text/plain` and `application/octet-stream` as having a registered [plugin](#).

EXAMPLE 31

One example of a plugin would be a PDF viewer that is instantiated in a [browsing context](#) when the user navigates to a PDF file. This would count as a plugin regardless of whether the party that implemented the PDF viewer component was the same as that which implemented the user agent itself. However, a PDF viewer application that launches separate from the user agent (as opposed to using the same interface) is not a plugin by this definition.

NOTE:

This specification does not define a mechanism for interacting with [plugins](#), as it is expected to be user-agent- and platform-specific. Some user agents might opt to support a plugin mechanism such as the Netscape Plugin API; others might use remote content converters or have built-in support for certain types. Indeed, this specification doesn't require user agents to support [plugins](#) at all. [\[NPAPI\]](#)

A plugin can be **secured** if it honors the semantics of the [sandbox](#) attribute.

EXAMPLE 32

For example, a secured plugin would prevent its contents from creating pop-up windows when the plugin is instantiated inside a sandboxed `<iframe>`.

⚠Warning! Browsers should take extreme care when interacting with external content intended for [plugins](#). When third-party software is run with the same privileges as the user agent itself, vulnerabilities in the third-party software become as dangerous as if they were vulnerabilities of the user agent itself.

Since different users having different sets of [plugins](#) provides a fingerprinting vector that increases the chances of users be-

ing uniquely identified, user agents are encouraged to support the exact same set of [plugins](#) for each user. 

§ 2.1.6. Character encodings

A **character encoding**, or just *encoding* where that is not ambiguous, is a defined way to convert between byte streams and Unicode strings, as defined in the WHATWG Encoding standard. An [encoding](#) has an **encoding name** and one or more **encoding labels**, referred to as the encoding's *name* and *labels* in the Encoding standard. [\[ENCODING\]](#)

A **UTF-16 encoding** is [UTF-16BE](#) or [UTF-16LE](#). [\[ENCODING\]](#)

An **ASCII-compatible encoding** is any [encoding](#) that is not a [UTF-16 encoding](#). [\[ENCODING\]](#)

NOTE:

Since support for encodings that are not defined in the WHATWG Encoding standard is prohibited, [UTF-16 encodings](#) are the only encodings that this specification needs to treat as not being [ASCII-compatible encodings](#).

The term **code unit** is used as defined in the Web IDL specification: a 16 bit unsigned integer, the smallest atomic component of a DOMString. (This is a narrower definition than the one used in Unicode, and is not the same as a *code point*.) [\[WEBIDL\]](#)

The term **Unicode code point** means a *Unicode scalar value* where possible, and an isolated surrogate code point when not. When a conformance requirement is defined in terms of characters or Unicode code points, a pair of [code units](#) consisting of a high surrogate followed by a low surrogate must be treated as the single code point represented by the surrogate pair, but isolated surrogates must each be treated as the single code point with the value of the surrogate. [\[UNICODE\]](#)

In this specification, the term **character**, when not qualified as *Unicode* character, is synonymous with the term [Unicode code point](#).

The term **Unicode character** is used to mean a *Unicode scalar value* (i.e. any Unicode code point that is not a surrogate code point). [\[UNICODE\]](#)

The **code-unit length** of a string is the number of [code units](#) in that string.

NOTE:

This complexity results from the historical decision to define the DOM API in terms of 16 bit (UTF-16) [code units](#), rather than in terms of [Unicode characters](#).

§ 2.2. Conformance requirements

All diagrams, examples, and notes in this specification are non-normative, as are all sections explicitly marked non-normative. Everything else in this specification is normative.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in the normative parts of this document are to be interpreted as described in RFC2119. The key word "OPTIONALLY" in the normative parts of this document is to be interpreted with the same normative meaning as "MAY" and "OPTIONAL". For readability, these words do not appear in all uppercase letters in this specification. [\[RFC2119\]](#)

Requirements phrased in the imperative as part of algorithms (such as "strip any leading space characters" or "return false and abort these steps") are to be interpreted with the meaning of the key word ("must", "should", "may", etc) used in introducing the algorithm.

EXAMPLE 33

For example, were the spec to say:

- To eat an orange, the user must:
1. Peel the orange.
 2. Separate each slice of the orange.
 3. Eat the orange slices.

...it would be equivalent to the following:

- To eat an orange:
1. The user must peel the orange.
 2. The user must separate each slice of the orange.
 3. The user must eat the orange slices.

Here the key word is "must".

The former (imperative) style is generally preferred in this specification for stylistic reasons.

Conformance requirements phrased as algorithms or specific steps may be implemented in any manner, so long as the end result is equivalent. (In particular, the algorithms defined in this specification are intended to be easy to follow, and not intended to be performant.)

§ 2.2.1. Conformance classes

This specification describes the conformance criteria for user agents (relevant to implementors) and documents (relevant to authors and authoring tool implementors).

Conforming documents are those that comply with all the conformance criteria for documents. For readability, some of these conformance requirements are phrased as conformance requirements on authors; such requirements are implicitly requirements on documents: by definition, all documents are assumed to have had an author. (In some cases, that author may itself be a user agent — such user agents are subject to additional rules, as explained below.)

EXAMPLE 34

For example, if a requirement states that "authors must not use the `foobar` element", it would imply that documents are not allowed to contain elements named `foobar`.

NOTE:

There is no implied relationship between document conformance requirements and implementation conformance requirements. User agents are not free to handle non-conformant documents as they please; the processing model described in this specification applies to implementations regardless of the conformity of the input documents.

User agents fall into several (overlapping) categories with different conformance requirements.

Web browsers and other interactive user agents

Web browsers that support [the XHTML syntax](#) must process elements and attributes from the [HTML namespace](#) found in XML documents as described in this specification, so that users can interact with them, unless the semantics of those elements have been overridden by other specifications.

EXAMPLE 35

A conforming XHTML processor would, upon finding an XHTML `<script>` element in an XML document, execute the script contained in that element. However, if the element is found within a transformation expressed in XSLT (assuming the user agent also supports XSLT), then the processor would instead treat the `<script>` element as an opaque element that forms part of the transform.

Web browsers that support [the HTML syntax](#) must process documents labeled with an [HTML MIME type](#) as described in this specification, so that users can interact with them.

User agents that support scripting must also be conforming implementations of the IDL fragments in this specification,

as described in the Web IDL specification. [\[WEBIDL\]](#)

NOTE:

Unless explicitly stated, specifications that override the semantics of HTML elements do not override the requirements on DOM objects representing those elements. For example, the `<script>` element in the example above would still implement the [HTMLScriptElement](#) interface.

Non-interactive presentation user agents

User agents that process HTML and XHTML documents purely to render non-interactive versions of them must comply to the same conformance criteria as Web browsers, except that they are exempt from requirements regarding user interaction.

NOTE:

Typical examples of non-interactive presentation user agents are printers (static user agents) and overhead displays (dynamic user agents). It is expected that most static non-interactive presentation user agents will also opt to [lack scripting support](#).

EXAMPLE 36

A non-interactive but dynamic presentation user agent would still execute scripts, allowing forms to be dynamically submitted, and so forth. However, since the concept of "focus" is irrelevant when the user cannot interact with the document, the user agent would not need to support any of the focus-related DOM APIs.

Visual user agents that support the suggested default rendering

User agents, whether interactive or not, may be designated (possibly as a user option) as [supporting the suggested default rendering](#) defined by this specification.

This is not required. In particular, even user agents that do implement the suggested default rendering are encouraged to offer settings that override this default to improve the experience for the user, e.g., changing the color contrast, using different focus styles, or otherwise making the experience more accessible and usable to the user.

User agents that are designated as [supporting the suggested default rendering](#) must, while so designated, implement the rules in [§10 Rendering](#). That section defines the behavior that user agents are *expected* to implement.

User agents with no scripting support

Implementations that do not support scripting (or which have their scripting features disabled entirely) are exempt from supporting the events and DOM interfaces mentioned in this specification. For the parts of this specification that are defined in terms of an events model or in terms of the DOM, such user agents must still act as if events and the DOM were supported.

NOTE:

Scripting can form an integral part of an application. Web browsers that do not support scripting, or that have scripting disabled, might be unable to fully convey the author's intent.

Conformance checkers

Conformance checkers must verify that a document conforms to the applicable conformance criteria described in this specification. Automated conformance checkers are exempt from detecting errors that require interpretation of the author's intent (for example, while a document is non-conforming if the content of a `<blockquote>` element is not a quote, conformance checkers running without the input of human judgement do not have to check that `<blockquote>` elements only contain quoted material).

Conformance checkers must check that the input document conforms when parsed without a [browsing context](#) (meaning that no scripts are run, and that the parser's [scripting flag](#) is disabled), and should also check that the input document conforms when parsed with a [browsing context](#) in which scripts execute, and that the scripts never cause non-conforming states to occur other than transiently during script execution itself. (This is only a "SHOULD" and not a "MUST" requirement because it has been proven to be impossible. [\[COMPUTABLE\]](#))

The term "HTML validator" can be used to refer to a conformance checker that itself conforms to the applicable requirements of this specification.

XML DTDs cannot express all the conformance requirements of this specification. Therefore, a validating XML processor and a DTD cannot constitute a conformance checker. Also, since neither of the two authoring formats defined in this specification are applications of SGML, a validating SGML system cannot constitute a conformance checker either.

To put it another way, there are three types of conformance criteria:

1. Criteria that can be expressed in a DTD.
2. Criteria that cannot be expressed by a DTD, but can still be checked by a machine.
3. Criteria that can only be checked by a human.

A conformance checker must check for the first two. A simple DTD-based validator only checks for the first class of errors and is therefore not a conforming conformance checker according to this specification.

Data mining tools

Applications and tools that process HTML and XHTML documents for reasons other than to either render the documents or check them for conformance should act in accordance with the semantics of the documents that they process.

EXAMPLE 37

A tool that generates [document outlines](#) but increases the nesting level for each paragraph and does not increase the nesting level for each section would not be conforming.

Authoring tools and markup generators

Authoring tools and markup generators must generate [conforming documents](#). Conformance criteria that apply to authors also apply to authoring tools, where appropriate.

Authoring tools are exempt from the strict requirements of using elements only for their specified purpose, but only to the extent that authoring tools are not yet able to determine author intent. However, authoring tools must not automatically misuse elements or encourage their users to do so.

EXAMPLE 38

For example, it is not conforming to use an `<address>` element for arbitrary contact information; that element can only be used for marking up contact information for the author of the document or section. However, since an authoring tool is likely unable to determine the difference, an authoring tool is exempt from that requirement. This does not mean, though, that authoring tools can use `<address>` elements for any block of italics text (for instance); it just means that the authoring tool doesn't have to verify, if a user inserts contact information for a section or something else.

NOTE:

In terms of conformance checking, an editor has to output documents that conform to the same extent that a conformance checker will verify.

When an authoring tool is used to edit a non-conforming document, it may preserve the conformance errors in sections of the document that were not edited during the editing session (i.e., an editing tool is allowed to round-trip erroneous content). However, an authoring tool must not claim that the output is conformant if errors have been so preserved.

Authoring tools are expected to come in two broad varieties: tools that work from structure or semantic data, and tools that work on a What-You-See-Is-What-You-Get media-specific editing basis (WYSIWYG).

The former is the preferred mechanism for tools that author HTML, since the structure in the source information can be used to make informed choices regarding which HTML elements and attributes are most appropriate.

However, WYSIWYG tools are legitimate. WYSIWYG tools should use elements they know are appropriate, and should not use elements that they do not know to be appropriate. This might in certain extreme cases mean limiting the use of flow elements to just a few elements, like `<div>`, `<b>`, `<i>`, and `<span>` and making liberal use of the `style` attribute.

All authoring tools, whether WYSIWYG or not, should make a best effort attempt at enabling users to create well-structured, semantically rich, media-independent content.

User agents may impose implementation-specific limits on otherwise unconstrained inputs, e.g., to prevent denial of service attacks, to guard against running out of memory, or to work around platform-specific limitations. 

For compatibility with existing content and prior specifications, this specification describes two authoring formats: one based on XML (referred to as [the XHTML syntax](#)), and one using a [custom format](#) inspired by SGML (referred to as [the HTML syntax](#)). Implementations must support at least one of these two formats, although supporting both is encouraged.

Some conformance requirements are phrased as requirements on elements, attributes, methods or objects. Such requirements fall into two categories: those describing content model restrictions, and those describing implementation behavior. Those in the former category are requirements on documents and authoring tools. Those in the second category are requirements on user agents. Similarly, some conformance requirements are phrased as requirements on authors; such requirements are to be interpreted as conformance requirements on the documents that authors produce. (In other words, this specification does not distinguish between conformance criteria on authors and conformance criteria on documents.)

§ 2.2.2. Dependencies

This specification relies on several other underlying specifications.

Unicode and Encoding

The Unicode character set is used to represent textual data, and the Encoding standard defines requirements around [character encodings](#). [\[UNICODE\]](#)

NOTE:

This specification [introduces terminology](#) based on the terms defined in those specifications, as described earlier.

The following terms are used as defined in the Encoding standard: [\[ENCODING\]](#)

- **Getting an encoding**
- **Get an output encoding**
- The generic [decode](#) algorithm which takes a byte stream and an encoding and returns a character stream
- The **UTF-8 decode** algorithm which takes a byte stream and returns a character stream, additionally stripping one leading UTF-8 Byte Order Mark (BOM), if any
- The **UTF-8 decode without BOM** algorithm which is identical to [UTF-8 decode](#) except that it does not strip one leading UTF-8 Byte Order Mark (BOM)
- The **UTF-8 decode without BOM or fail** algorithm which is identical to [UTF-8 decode without BOM](#) except that it returns failure upon encountering an error
- The [encode](#) algorithm which takes a character stream and an encoding and returns a byte stream
- The **UTF-8 encode** algorithm which takes a character stream and returns a byte stream.

XML and related specifications

Implementations that support [the XHTML syntax](#) must support some version of XML, as well as its corresponding namespaces specification, because that syntax uses an XML serialization with namespaces. [\[XML\]](#) [\[XML-NAMES\]](#)

The attribute with the tag name [xml:space](#) in the [XML namespace](#) is defined by the XML specification. [\[XML\]](#)

This specification also references the `<?xml-stylesheet?>` processing instruction, defined in the *Associating Style Sheets with XML documents* specification. [\[XML-stylesheet\]](#)

This specification also non-normatively mentions the `XSLTProcessor` interface and its `transformToFragment()` and `transformToDocument()` methods.

URLs

The following terms are defined in the WHATWG URL standard: [\[URL\]](#)

- [host](#)

- [domain](#)
- [URL](#)
- [Origin](#) of URLs
- [Absolute URL](#)
- [Relative URL](#)
- [Relative schemes](#)
- The [URL parser](#) and [basic URL parser](#) as well as these parser states:
 - [scheme start state](#)
 - [host state](#)
 - [hostname state](#)
 - [port state](#)
 - [path start state](#)
 - [query state](#)
 - [fragment state](#)
- [URL record](#), as well as its individual components:
 - [scheme](#)
 - [username](#)
 - [password](#)
 - [host](#)
 - [port](#)
 - [path](#)
 - [query](#)
 - [fragment](#)
- A [network scheme](#)
- The [URL serializer](#)
- The [host parser](#)
- The [host serializer](#)
- [Host equals](#)
- [serialize an integer](#)
- [Default encode set](#)
- [Percent encode](#)
- [UTF-8 percent encode](#)
- [Percent decode](#)
- [set the username](#)
- [set the password](#)

- The [domain to ASCII](#) algorithm
- The [domain to Unicode](#) algorithm
- [non-relative flag](#)
- [Parse errors](#) from the [URL parser](#)

A number of schemes and protocols are referenced by this specification also:

- The [about:](#) scheme [\[RFC6694\]](#)
- The [blob:](#) scheme [\[FILEAPI\]](#)
- The [data:](#) scheme [\[RFC2397\]](#)
- The [http:](#) scheme [\[HTTP\]](#)
- The [https:](#) scheme [\[HTTP\]](#)
- The [mailto:](#) scheme [\[RFC6068\]](#)
- The [sms:](#) scheme [\[RFC5724\]](#)
- The [urn:](#) scheme [\[URN\]](#)

HTTP and related specifications

The following terms are defined in the HTTP specifications: [\[HTTP\]](#)

- [Accept](#) header
- [Accept-Language](#) header
- [Cache-Control](#) header
- [Content-Disposition](#) header
- [Content-Language](#) header
- [Content-Length](#) header
- [Last-Modified](#) header
- [Referer](#) header

The following terms are defined in the Cookie specification: [\[COOKIES\]](#)

- **cookie-string**
- [receives a set-cookie-string](#)
- [Cookie header](#)

The following term is defined in the Web Linking specification: [\[RFC5988\]](#)

- [Link header](#)

Fetch

The following terms are defined in the WHATWG Fetch standard: [\[FETCH\]](#)

- [about:blank](#)
- [HTTPS state value](#)
- [referrer policy](#)
- [CORS protocol](#)
- [default User-Agent value](#)

- [extract a MIME type](#)
- [fetch](#)
- [ok status](#)
- [Origin header](#)
- [process response](#)
- [set](#)
- [terminate](#)
- the [RequestCredentials](#) enumeration
- [response](#) and its associated:
 - [type](#)
 - [url](#)
 - [url list](#)
 - [status](#)
 - [header list](#)
 - [body](#)
 - [internal response](#)
 - [CSP list](#)
 - [HTTPS state](#)
- [request](#) and its associated:
 - [url](#)
 - [method](#)
 - [header list](#)
 - [body](#)
 - [client](#)
 - [target browsing context](#)
 - [initiator](#)
 - [type](#)
 - [destination](#)
 - [origin](#)
 - [omit-Origin-header flag](#)
 - [same-origin data-URL flag](#)
 - [referrer](#)
 - [synchronous flag](#)
 - [mode](#)
 - [credentials mode](#)

- [use-URL-credentials flag](#)
- [unsafe-request flag](#)
- [cache mode](#)
- [redirect mode](#)
- [cryptographic nonce metadata](#)
- [parser metadata](#)

Web IDL

The IDL fragments in this specification must be interpreted as required for conforming IDL fragments, as described in the Web IDL specification. [\[WEBIDL\]](#)

The following terms are defined in the Web IDL specification:

- [array index property name](#)
- [supported property indices](#)
- [Determine the value of an indexed property](#)
- [Support named properties](#)
- [Supported property names](#)
- [Determine the value of a named property](#)
- [perform a security check](#)
- [Platform object](#)
- [Global environment associated with](#) a platform object
- [Read only](#) (when applied to arrays)
- [Callback this value](#)
- [Converting](#) between WebIDL types and JS types
- [invoke the Web IDL callback function](#)

The Web IDL specification also defines the following types that are used in Web IDL fragments in this specification:

- [ArrayBufferView](#)
- [‘boolean’](#)
- [‘DOMString’](#)
- [‘USVString’](#)
- [‘double’](#)
- [‘Error’](#)
- [Function](#)
- [‘long’](#)
- [object](#)
- [‘unrestricted double’](#)
- [‘unsigned long’](#)

The term **throw** in this specification is used as defined in the WebIDL specification. The following exception names

are defined by WebIDL and used by this specification:

- [IndexSizeError](#)
- [HierarchyRequestError](#)
- [InvalidCharacterError](#)
- [NotFoundError](#)
- [NotSupportedError](#)
- [InvalidStateError](#)
- [SyntaxError](#)
- [InvalidAccessError](#)
- [SecurityError](#)
- [NetworkError](#)
- [QuotaExceededError](#)
- [TimeoutError](#)
- [DataCloneError](#)

When this specification requires a user agent to **create a [Date](#) object** representing a particular time (which could be the special value Not-a-Number), the milliseconds component of that time, if any, must be truncated to an integer, and the time value of the newly created [Date](#) object must represent the resulting truncated time.

EXAMPLE 39

For instance, given the time 23045 millionths of a second after 01:00 UTC on January 1st 2000, i.e., the time 2000-01-01T00:00:00.023045Z, then the [Date](#) object created representing that time would represent the same time as that created representing the time 2000-01-01T00:00:00.023Z, 45 millionths earlier. If the given time is NaN, then the result is a [Date](#) object that represents a time value NaN (indicating that the object does not represent a specific instant of time).

JavaScript

Some parts of the language described by this specification only support JavaScript as the underlying scripting language. [\[ECMA-262\]](#)

NOTE:

The term "JavaScript" is used to refer to ECMA262, rather than the official term ECMAScript, since the term JavaScript is more widely known. Similarly, the [MIME type](#) used to refer to JavaScript in this specification is text/javascript, since that is the most commonly used type, [despite it being an officially obsoleted type](#) according to RFC 4329. [\[RFC4329\]](#)

The following terms are defined in the JavaScript specification and used in this specification [\[ECMA-262\]](#):

- [automatic semicolon insertion](#)
- [early error](#)
- [Directive Prologue](#)
- [JavaScript execution context](#)
- [JavaScript execution context stack](#)
- [running JavaScript execution context](#)
- [JavaScript realm](#)

- The [current Realm Record](#)
- Use Strict Directive
- [Well-Known Symbols](#), including:
 - [@@hasInstance](#)
 - [@@isConcatSpreadable](#)
 - [@@toPrimitive](#)
 - [@@toStringTag](#)
- [Well-Known Intrinsic Objects](#), including:
 - [%ArrayBuffer%](#)
 - [%ArrayPrototype%](#)
 - [%ObjProto toString%](#)
 - [%ObjProto valueOf%](#)
- The [FunctionBody](#) production
- The [Pattern](#) production
- The [Script](#) production
- The [Type](#) notation
- The [List](#) and [Record](#) specification types
- The [Property Descriptor](#) specification type
- The [ArrayCreate](#) abstract operation
- The [Call](#) abstract operation
- The [CloneArrayBuffer](#) abstract operation
- The [Construct](#) abstract operation
- The [CreateDataProperty](#) abstract operation
- The [DetachArrayBuffer](#) abstract operation
- The [EnqueueJob](#) abstract operation
- The [FunctionCreate](#) abstract operation
- The [Get](#) abstract operation
- The [GetActiveScriptOrModule](#) abstract operation
- The [GetFunctionRealm](#) abstract operation
- The [HasOwnProperty](#) abstract operation
- The [HostEnsureCanCompileStrings](#) abstract operation
- The [HostPromiseRejectionTracker](#) abstract operation
- The [InitializeHostDefinedRealm](#) abstract operation
- The [IsAccessorDescriptor](#) abstract operation
- The [IsCallable](#) abstract operation

- The [IsConstructor](#) abstract operation
- The [IsDataDescriptor](#) abstract operation
- The [IsDetachedBuffer](#) abstract operation
- The [NewObjectEnvironment](#) abstract operation
- The [OrdinaryGetPrototypeOf](#) abstract operation
- The [OrdinarySetPrototypeOf](#) abstract operation
- The [OrdinaryIsExtensible](#) abstract operation
- The [OrdinaryPreventExtensions](#) abstract operation
- The [OrdinaryGetOwnProperty](#) abstract operation
- The [OrdinaryDefineOwnProperty](#) abstract operation
- The [OrdinaryGet](#) abstract operation
- The [OrdinarySet](#) abstract operation
- The [OrdinaryDelete](#) abstract operation
- The [OrdinaryOwnPropertyKeys](#) abstract operation
- The [ParseScript](#) abstract operation
- The [RunJobs](#) abstract operation
- The [SameValue](#) abstract operation
- The [ScriptEvaluation](#) abstract operation
- The [ToBoolean](#) abstract operation
- The [ToString](#) abstract operation
- The [ToUint32](#) abstract operation
- The [TypedArrayCreate](#) abstract operation
- The [Abstract Equality Comparison](#) algorithm
- The [Strict Equality Comparison](#) algorithm
- The [ArrayBuffer](#) object
- The [Date](#) object
- The [SyntaxError](#) object
- The [TypeError](#) object
- The [RangeError](#) object
- The [RegExp](#) object
- The [typeof](#) operator
- The [TypedArray Constructors](#) table

DOM

The Document Object Model (DOM) is a representation — a model — of a document and its content. The DOM is not just an API; the conformance criteria of HTML implementations are defined, in this specification, in terms of operations on the DOM. [\[DOM\]](#)

Implementations must support DOM and the events defined in UI Events, because this specification is defined in terms of the DOM, and some of the features are defined as extensions to the DOM interfaces. [\[DOM\]](#) [\[UIEVENTS\]](#)

In particular, the following features are defined in the DOM specification: [\[DOM\]](#)

- [Attr](#) interface
- [Comment](#) interface
- [DOMImplementation](#) interface
- [Document](#) interface
- [XMLDocument](#) interface
- [DocumentFragment](#) interface
- [DocumentType](#) interface
- [DOMException](#) interface
- [ChildNode](#) interface
- [Element](#) interface
- [Node](#) interface
- [NodeList](#) interface
- [ProcessingInstruction](#) interface
- [Text](#) interface
- [HTMLCollection](#) interface
- [item\(\)](#) method
- The terms [collections](#) and [represented by the collection](#)
- [DOMTokenList](#) interface
- [createDocument\(\)](#) method
- [createHTMLDocument\(\)](#) method
- [createElement\(\)](#) method
- [createElementNS\(\)](#) method
- [getElementById\(\)](#) method
- [getElementsByClassName\(\)](#) method
- [insertBefore\(\)](#) method
- [appendChild\(\)](#) method
- [cloneNode\(\)](#) method
- [importNode\(\)](#) method
- [childNodes](#) attribute
- [localName](#) attribute
- [parentNode](#) attribute
- [namespaceURI](#) attribute

- [tagName](#) attribute
- [id](#) attribute
- [textContent](#) attribute
- The [insert](#), [append](#), [remove](#), [replace](#), and [adopt](#) algorithms for nodes
- The [insertion steps](#), [removing steps](#), and [adopting steps](#) hooks
- The [attribute list](#) concept.
- The [data](#) of a text node.
- [Event](#) interface
- [EventTarget](#) interface
- [EventInit](#) dictionary type
- [target](#) attribute
- [currentTarget](#) attribute
- [isTrusted](#) attribute
- [initEvent\(\)](#) method
- [addEventListener\(\)](#) method
- The [type](#) of an event
- The concept of an [event listener](#) and the [event listeners](#) associated with an [EventTarget](#)
- The concept of a **target override**
- The concept of a regular **event parent** and a **cross-boundary event parent**
- The [encoding](#) (herein the *character encoding*) and [content type](#) of a [Document](#)
- The distinction between [XML documents](#) and [HTML documents](#)
- The terms **quirks mode**, **limited-quirks mode**, and **no-quirks mode**
- The algorithm to [clone](#) a [Node](#), and the concept of [cloning steps](#) used by that algorithm
- The concept of **base URL change steps** and the definition of what happens when an element is **affected by a base URL change**
- The concept of an element's [unique identifier \(ID\)](#)
- The term **supported tokens**
- The concept of a DOM [range](#), and the terms [start](#), [end](#), and [boundary point](#) as applied to ranges.
- [MutationObserver](#) interface and **mutation observers** in general

The term [throw](#) in this specification is used as defined in the DOM specification. The following [DOMException](#) types are defined in the DOM specification: [\[DOM\]](#)

- [IndexSizeError](#)
- [HierarchyRequestError](#)
- [WrongDocumentError](#)
- [InvalidCharacterError](#)
- [NoModificationAllowedError](#)

- [NotFoundError](#)
- [NotSupportedError](#)
- [InvalidStateError](#)
- [SyntaxError](#)
- [InvalidModificationError](#)
- [NamespaceError](#)
- [InvalidAccessError](#)
- [SecurityError](#)
- [NetworkError](#)
- [AbortError](#)
- [URLMismatchError](#)
- [QuotaExceededError](#)
- [TimeoutError](#)
- [InvalidNodeTypeError](#)
- [DataCloneError](#)

EXAMPLE 40

For example, to *throw* a [TimeoutError](#) exception, a user agent would construct a [DOMException](#) object whose type was the string "TimeoutError" (and whose code was the number 23, for legacy reasons) and actually throw that object as an exception.

The following features are defined in the UI Events specification: [\[UIEVENTS\]](#)

- [MouseEvent](#) interface and the following interface members:
 - The [relatedTarget](#) attribute
 - The [button](#) attribute
 - The [ctrlKey](#) attribute
 - The [shiftKey](#) attribute
 - The [altKey](#) attribute
 - The [metaKey](#) attribute
 - The [getModifierState\(\)](#) method
- [MouseEventInit](#) dictionary type
- The [FocusEvent](#) interface and its [relatedTarget](#) attribute
- The [UIEvent](#) interface's [view](#) and [detail](#) attributes
- [click](#) event
- [dblclick](#) event
- [mousedown](#) event
- [mouseenter](#) event
- [mouseleave](#) event

- [mousemove](#) event
- [mouseout](#) event
- [mouseover](#) event
- [mouseup](#) event
- [wheel](#) event
- [keydown](#) event
- [keyup](#) event
- [keypress](#) event

The following features are defined in the Touch Events specification: [\[TOUCH-EVENTS\]](#)

- [Touch](#) interface
- [Touch point](#) concept

This specification sometimes uses the term **name** to refer to the event's type; as in, "an event named `click`" or "if the event name is `keypress`". The terms "name" and "type" for events are synonymous.

The following features are defined in the DOM Parsing and Serialization specification: [\[DOM-Parsing\]](#)

- [innerHTML](#)
- [outerHTML](#)

The [Selection](#) interface is defined in the *Selection API* specification. [\[SELECTION-API\]](#)

NOTE:

User agents are also encouraged to implement the features described in the *HTML Editing APIs* and *UndoManager* and *DOM Transaction* specifications. [\[EDITING\]](#) [\[UNDO\]](#)

The following parts of the Fullscreen specification are referenced from this specification, in part to define how the Fullscreen API interacts with the sandboxing features in HTML: [\[FULLSCREEN\]](#)

- The [top layer](#) concept
- [requestFullscreen\(\)](#)
- The [fullscreen enabled flag](#)
- The [fully exit fullscreen](#) algorithm

The *High Resolution Time* specification provides the [DOMHighResTimeStamp](#) typedef and the [Performance](#) object's [now\(\)](#) method. [\[HR-TIME-2\]](#)

File API

This specification uses the following features defined in the File API specification: [\[FILEAPI\]](#)

- [Blob](#) interface
- [File](#) interface
- [FileList](#) interface
- The concept of a [closed Blob](#)
- [Blob.type](#)
- The concept of **read errors**

Media Source Extensions

The following terms are defined in the Media Source Extensions specification: [\[MEDIA-SOURCE\]](#)

- **Detaching from a media element**
- [MediaSource](#) [\[MEDIA-SOURCE\]](#)

Media Capture and Streams

The following term is defined in the Media Capture and Streams specification: [\[MEDIACAPTURE-STREAMS\]](#)

- [MediaStream](#) [\[MEDIACAPTURE-STREAMS\]](#)

XMLHttpRequest

This specification references the XMLHttpRequest specification to describe how the two specifications interact. The following features and terms are defined in the XMLHttpRequest specification: [\[XHR\]](#)

- [XMLHttpRequest](#) interface
- [XMLHttpRequest.responseXML](#) attribute

ProgressEvent

This specification references the Progress Events specification to describe how the two specifications interact and to use its [ProgressEvent](#) feature. The following features are defined in the Progress Events specification: [\[PROGRESS-EVENTS\]](#)

- [ProgressEvent](#) interface
- [ProgressEvent.lengthComputable](#) attribute
- [ProgressEvent.loaded](#) attribute
- [ProgressEvent.total](#) attribute
- [Fire a progress event named *e*](#)

Server-Sent Events

This specification references [EventSource](#) which is specified in the Server-Sent Events specification [\[EVENTSOURCE\]](#)

Media Queries

Implementations must support the Media Queries language. [\[MEDIAQ\]](#)

CSS modules

While support for CSS as a whole is not required of implementations of this specification (though it is encouraged, at least for Web browsers), some features are defined in terms of specific CSS requirements.

In particular, some features require that a string be **parsed as a CSS <color> value**. When parsing a CSS value, user agents are required by the CSS specifications to apply some error handling rules. These apply to this specification also. [\[CSS3COLOR\]](#) [\[CSS-2015\]](#)

EXAMPLE 41

For example, user agents are required to close all open constructs upon finding the end of a style sheet unexpectedly. Thus, when parsing the string "rgb(0,0,0" (with a missing close-parenthesis) for a color value, the close parenthesis is implied by this error handling rule, and a value is obtained (the color **'black'**). However, the similar construct "rgb(0,0," (with both a missing parenthesis and a missing "blue" value) cannot be parsed, as closing the open construct does not result in a viable value.

The following terms and features are defined in the CSS specification: [\[CSS-2015\]](#)

- **viewport**
- **replaced element**
- **intrinsic dimensions**

The term **named color** is defined in the CSS Color specification. [\[CSS3-COLOR\]](#)

The terms **intrinsic width** and **intrinsic height** refer to the width dimension and the height dimension, respectively, of [intrinsic dimensions](#).

The term **provides a paint source** is used as defined in the *CSS Image Values and Replaced Content* specification to define the interaction of certain HTML elements with the CSS 'element()' function. [\[CSS3-IMAGES\]](#)

The term **default object size** is also defined in the *CSS Image Values and Replaced Content* specification. [\[CSS3-IMAGES\]](#)

Implementations that support scripting must support the CSS Object Model. The following features and terms are defined in the CSSOM specifications: [\[CSSOM\]](#) [\[CSSOM-VIEW\]](#)

- [Screen](#)
- [LinkStyle](#)
- [CSSStyleDeclaration](#)
- [cssText](#) attribute of [CSSStyleDeclaration](#)
- [StyleSheet](#)
- [create a CSS style sheet](#)
- [remove a CSS style sheet](#)
- [associated CSS style sheet](#)
- [CSS style sheets](#) and their properties: [type](#), [location](#), [parent CSS style sheet](#), [owner node](#), [owner CSS rule](#), [media](#), [title](#), [alternate flag](#), [disabled flag](#), [CSS rules](#), [origin-clean flag](#)
- [Alternative style sheet sets](#) and the [preferred style sheet set](#)
- [Serializing a CSS value](#)
- [run the resize steps](#)
- [run the scroll steps](#)
- [evaluate media queries and report changes](#)
- [Scroll an element into view](#)
- [Scroll to the beginning of the document](#)
- The [resize](#) event
- The [scroll](#) event
- The [features](#) argument of `window.open`

The following features and terms are defined in the *CSS Syntax* specifications: [\[CSS-SYNTAX-3\]](#)

- [Parse a comma-separated list of component values](#)
- [component value](#)
- [environment encoding](#)
- [‘<whitespace-token>’](#)

The feature [‘<length>’](#) is defined in the *CSS Values and Units* specification. [\[CSS-VALUES\]](#)

The term [CSS styling attribute](#) is defined in the *CSS Style Attributes* specification. [\[CSS-STYLE-ATTR\]](#)

The `CanvasRenderingContext2D` object’s use of fonts depends on the features described in the *CSS Fonts* and *Font*

Loading specifications, including in particular [FontFace](#) objects and the [font source](#) concept. [\[CSS-FONTS-3\]](#)
[\[CSS-FONT-LOADING-3\]](#)

The following interface is defined in the Geometry Interfaces Module specification: [\[GEOMETRY-1\]](#)

- [DOMMatrix](#) interface

SVG

The `CanvasRenderingContext2D` object's use of fonts depends on the features described in the CSS *Fonts* and *Font Loading* specifications, including in particular [FontFace](#) objects and the [font source](#) concept. [\[CSS-FONTS-3\]](#)
[\[CSS-FONT-LOADING-3\]](#)

The following interface is defined in the SVG specification: [\[SVG11\]](#)

- [SVGMatrix](#)

WebGL

The following interface is defined in the WebGL specification: [\[WEBGL\]](#)

- [WebGLRenderingContext](#)

WebVTT

Implementations may support **WebVTT** as a text track format for subtitles, captions, chapter titles, metadata, etc, for media resources. [\[WEBVTT\]](#)

The following terms, used in this specification, are defined in the WebVTT specification:

- **WebVTT file**
- **WebVTT file using cue text**
- **WebVTT file using chapter title text**
- **WebVTT file using only nested cues**
- **WebVTT parser**
- **The rules for updating the display of WebVTT text tracks**
- **The rules for interpreting WebVTT cue text**
- **The WebVTT text track cue writing direction**

The WebSocket protocol

The following terms are defined in the WebSocket protocol specification: [\[RFC6455\]](#)

- **establish a WebSocket connection**
- **the WebSocket connection is established**
- **validate the server's response**
- **extensions in use**
- **subprotocol in use**
- **headers to send appropriate cookies**
- **cookies set during the server's opening handshake**
- **a WebSocket message has been received**
- **send a WebSocket Message**
- **fail the WebSocket connection**
- **close the WebSocket connection**

- **start the WebSocket closing handshake**
- **the WebSocket closing handshake is started**
- **the WebSocket connection is closed** (possibly *cleanly*)
- **the WebSocket connection close code**
- **the WebSocket connection close reason**
- **Sec-WebSocket-Protocol** field

ARIA

The **role** attribute is defined in the ARIA specification, as are the following roles: [\[WAI-ARIA\]](#)

- [‘alert’](#)
- [‘alertdialog’](#)
- [‘application’](#)
- [‘article’](#)
- [‘banner’](#)
- [‘button’](#)
- [‘checkbox’](#)
- [‘columnheader’](#)
- [‘combobox’](#)
- [‘complementary’](#)
- [‘contentinfo’](#)
- [‘dialog’](#)
- [‘directory’](#)
- [‘document’](#)
- [‘grid’](#)
- [‘gridcell’](#)
- [‘group’](#)
- [‘heading’](#)
- [‘img’](#)
- [‘link’](#)
- [‘list’](#)
- [‘listbox’](#)
- [‘listitem’](#)
- [‘log’](#)
- [‘main’](#)
- [‘marquee’](#)
- [‘menu’](#)

- [‘menubar’](#)
- [‘menuitem’](#)
- [‘menuitemcheckbox’](#)
- [‘menuitemradio’](#)
- [‘navigation’](#)
- [‘note’](#)
- [‘option’](#)
- [‘presentation’](#)
- [‘progressbar’](#)
- [‘radio’](#)
- [‘region’](#)
- [‘row’](#)
- [‘rowgroup’](#)
- [‘rowheader’](#)
- [‘search’](#)
- [‘separator’](#)
- [‘slider’](#)
- [‘spinbutton’](#)
- [‘status’](#)
- [‘tab’](#)
- [‘tablist’](#)
- [‘textbox’](#)
- [‘toolbar’](#)
- [‘tree’](#)
- [‘treeitem’](#)

In addition, the following **aria-*** content attributes are defined in the ARIA specification: [\[WAI-ARIA\]](#)

- **aria-checked**
- **aria-describedby**
- **aria-disabled**
- **aria-expanded**
- **aria-hidden**
- **aria-invalid**
- **aria-label**
- **aria-level**
- **aria-multiline**

- **aria-multiselectable**
- **aria-owns**
- **aria-readonly**
- **aria-required**
- **aria-selected**
- **aria-sort**
- **aria-valuemax**
- **aria-valuemin**
- **aria-valuenow**

Content Security Policy

The following terms are defined in *Content Security Policy*: [\[CSP3\]](#)

- [Content Security Policy](#)
- [Content Security Policy directive](#)
- The [Content Security Policy syntax](#)
- [enforce the policy](#)
- The [parse a serialized Content Security Policy](#) algorithm
- The [Initialize a global object's CSP list](#) algorithm
- The [Initialize a Document's CSP list](#) algorithm
- The [Should element's inline behavior be blocked by Content Security Policy?](#) algorithm
- The [report-uri](#), [frame-ancestors](#), and [sandbox directives](#)
- The [EnsureCSPDoesNotBlockStringCompilation](#) abstract algorithm
- The [Is base allowed for Document?](#) algorithm

The following terms are defined in *Content Security Policy: Document Features*

- The [frame-ancestors directive](#)
- The [sandbox directive](#)

Service Workers

The following terms are defined in *Service Workers*: [\[SERVICE-WORKERS\]](#)

- **match service worker registration**

This specification does not *require* support of any particular network protocol, style sheet language, scripting language, or any of the DOM specifications beyond those required in the list above. However, the language described by this specification is biased towards CSS as the styling language, JavaScript as the scripting language, and HTTP as the network protocol, and several features assume that those languages and protocols are in use.

A user agent that implements the HTTP protocol must implement the Web Origin Concept specification and the HTTP State Management Mechanism specification (Cookies) as well. [\[HTTP\]](#) [\[ORIGIN\]](#) [\[COOKIES\]](#)

NOTE:

This specification might have certain additional requirements on character encodings, image formats, audio formats, and video formats in the respective sections.

§ 2.2.3. Extensibility

Vendor-specific proprietary user agent extensions to this specification are strongly discouraged. Documents must not use such extensions, as doing so reduces interoperability and fragments the user base, allowing only users of specific user agents to access the content in question.

If such extensions are nonetheless needed, e.g., for experimental purposes, then vendors are strongly urged to use one of the following extension mechanisms:

- For markup-level features that can be limited to the XML serialization and need not be supported in the HTML serialization, vendors should use the namespace mechanism to define custom namespaces in which the non-standard elements and attributes are supported.
- For markup-level features that are intended for use with [the HTML syntax](#), extensions should be limited to new attributes of the form "x-*vendor*-*feature*", where *vendor* is a short string that identifies the vendor responsible for the extension, and *feature* is the name of the feature. New element names should not be created. Using attributes for such extensions exclusively allows extensions from multiple vendors to co-exist on the same element, which would not be possible with elements. Using the "x-*vendor*-*feature*" form allows extensions to be made without risk of conflicting with future additions to the specification.

EXAMPLE 42

For instance, a browser named "FerretBrowser" could use "ferret" as a vendor prefix, while a browser named "Mellblom Browser" could use "mb". If both of these browsers invented extensions that turned elements into scratch-and-sniff areas, an author experimenting with these features could write:

```
<p>This smells of lemons!
<span x-ferret-smellovision x-ferret-smellcode="LEM01"
x-mb-outputsmell x-mb-smell="lemon juice"></span></p>
```

Attribute names beginning with the two characters "x-" are reserved for user agent use and are guaranteed to never be formally added to the HTML language. For flexibility, attributes names containing underscores (the U+005F LOW LINE character) are also reserved for experimental purposes and are guaranteed to never be formally added to the HTML language.

NOTE:

Pages that use such attributes are by definition non-conforming.

For DOM extensions, e.g., new methods and IDL attributes, the new members should be prefixed by vendor-specific strings to prevent clashes with future versions of this specification.

For events, experimental event types should be prefixed with vendor-specific strings.

EXAMPLE 43

For example, if a user agent called "Pleasold" were to add an event to indicate when the user is going up in an elevator, it could use the prefix "pleasold" and thus name the event "pleasoldgoingup", possibly with an event handler attribute named "onpleasoldgoingup".

All extensions must be defined so that the use of extensions neither contradicts nor causes the non-conformance of functionality defined in the specification.

EXAMPLE 44

For example, while strongly discouraged from doing so, an implementation "Foo Browser" could add a new IDL attribute "fooTypeTime" to a control's DOM interface that returned the time it took the user to select the current value of a control (say). On the other hand, defining a new control that appears in a form's `elements` array would be in violation of the above requirement, as it would violate the definition of `elements` given in this specification.

When adding new [reflecting](#) IDL attributes corresponding to content attributes of the form "x-*vendor*-*feature*", the IDL attribute should be named "*vendor* *Feature*" (i.e., the "x" is dropped from the IDL attribute's name).



When vendor-neutral extensions to this specification are needed, either this specification can be updated accordingly, or an extension specification can be written that overrides the requirements in this specification. When someone applying this specification to their activities decides that they will recognize the requirements of such an extension specification, it becomes an [applicable specification](#) for the purposes of conformance requirements in this specification.

NOTE:

Someone could write a specification that defines any arbitrary byte stream as conforming, and then claim that their random junk is conforming. However, that does not mean that their random junk actually is conforming for everyone's purposes: if someone else decides that the specification does not apply to their work, then they can quite legitimately say that the aforementioned random junk is just that, junk, and not conforming at all. As far as conformance goes, what matters in a particular community is what that community agrees is applicable.

applicable specification.

The conformance terminology for documents depends on the nature of the changes introduced by such applicable specifications, and on the content and intended interpretation of the document. Applicable specifications MAY define new document content (e.g., a foobar element), MAY prohibit certain otherwise conforming content (e.g., prohibit use of `<table>s`), or MAY change the semantics, DOM mappings, or other processing rules for content defined in this specification. Whether a document is or is not a [conforming HTML document](#) does not depend on the use of applicable specifications: if the syntax and semantics of a given [conforming HTML document](#) is unchanged by the use of applicable specification(s), then that document remains a [conforming HTML document](#). If the semantics or processing of a given (otherwise conforming) document is changed by use of applicable specification(s), then it is not a [conforming HTML document](#). For such cases, the applicable specifications SHOULD define conformance terminology.

NOTE:

As a suggested but not required convention, such specifications might define conformance terminology such as: "Conforming HTML+XXX document", where XXX is a short name for the applicable specification. (Example: "Conforming HTML+AutomotiveExtensions document").

NOTE:

a consequence of the rule given above is that certain syntactically correct HTML documents may not be [conforming HTML documents](#) in the presence of applicable specifications. (Example: the applicable specification defines `<table>` to be a piece of furniture — a document written to that specification and containing a `<table>` element is NOT a [conforming HTML document](#), even if the element happens to be syntactically correct HTML.)



User agents must treat elements and attributes that they do not understand as semantically neutral; leaving them in the DOM (for DOM processors), and styling them according to CSS (for CSS processors), but not inferring any meaning from them.

When support for a feature is disabled (e.g., as an emergency measure to mitigate a security problem, or to aid in development, or for performance reasons), user agents must act as if they had no support for the feature whatsoever, and as if the feature was not mentioned in this specification. For example, if a particular feature is accessed via an attribute in a Web IDL interface, the attribute itself would be omitted from the objects that implement that interface — leaving the attribute on the object but making it return null or throw an exception is insufficient.

§ 2.2.4. Interactions with XPath and XSLT

Implementations of XPath 1.0 that operate on [HTML documents](#) parsed or created in the manners described in this specification (e.g., as part of the document `.evaluate()` API) must act as if the following edit was applied to the XPath 1.0 specification.

First, remove this paragraph:

A [QName](#) in the node test is expanded into an [expanded-name](#) using the namespace declarations from the expression context. This is the same way expansion is done for element type names in start and end-tags except that the default namespace declared with `xmlns` is not used: if the [QName](#) does not have a prefix, then the namespace URI is null (this is the same way attribute names are expanded). It is an error if the [QName](#) has a prefix for which there is no namespace declaration in the expression context.

Then, insert in its place the following:

A [QName](#) in the node test is expanded into an [expanded-name](#) using the namespace declarations from the expression context. If the [QName](#) has a prefix, then there must be a namespace declaration for this prefix in the expression context, and the corresponding namespace URI is the one that is associated with this prefix. It is an error if the [QName](#) has a prefix for which there is no namespace declaration in the expression context.

If the [QName](#) has no prefix and the principal node type of the axis is element, then the default element namespace is used. Otherwise if the [QName](#) has no prefix, the namespace URI is null. The default element namespace is a member of the context for the XPath expression. The value of the default element namespace when executing an XPath expression through the DOM3 XPath API is determined in the following way:

1. If the context node is from an HTML DOM, the default element namespace is "[http://www.w3.org/1999/xhtml](#)".
2. Otherwise, the default element namespace URI is null.

NOTE:

This is equivalent to adding the default element namespace feature of XPath 2.0 to XPath 1.0, and using the HTML namespace as the default element namespace for HTML documents. It is motivated by the desire to have implementations be compatible with legacy HTML content while still supporting the changes that this specification introduces to HTML regarding the namespace used for HTML elements, and by the desire to use XPath 1.0 rather than XPath 2.0.

NOTE:

This change is a [willful violation](#) of the XPath 1.0 specification, motivated by desire to have implementations be compatible with legacy content while still supporting the changes that this specification introduces to HTML regarding which namespace is used for HTML elements. [\[XPATH\]](#)



XSLT 1.0 processors outputting to a DOM when the output method is "html" (either explicitly or via the defaulting rule in XSLT 1.0) are affected as follows:

If the transformation program outputs an element in no namespace, the processor must, prior to constructing the corresponding DOM element node, change the namespace of the element to the [HTML namespace](#), [ASCII-lowercase](#) the element's local name, and [ASCII-lowercase](#) the names of any non-namespaced attributes on the element.

NOTE:

This requirement is a [willful violation](#) of the XSLT 1.0 specification, required because this specification changes the namespaces and case-sensitivity rules of HTML in a manner that would otherwise be incompatible with DOM-based XSLT transformations. (Processors that serialize the output are unaffected.) [\[XSLT\]](#)



This specification does not specify precisely how XSLT processing interacts with the [HTML parser](#) infrastructure (for example, whether an XSLT processor acts as if it puts any elements into a [stack of open elements](#)). However, XSLT processors must [stop parsing](#) if they successfully complete, and must set the [current document readiness](#) first to "interactive" and then to "complete" if they are aborted.



This specification does not specify how XSLT interacts with the [navigation](#) algorithm, how it fits in with the [event loop](#), nor

how error pages are to be handled (e.g., whether XSLT errors are to replace an incremental XSLT output, or are rendered in-line, etc).

NOTE:

There are also additional non-normative comments regarding the interaction of XSLT and HTML [in the script element section](#), and of XSLT, XPath, and HTML [in the template element section](#).

§ 2.3. Case-sensitivity and string comparison

Comparing two strings in a **case-sensitive** manner means comparing them exactly, code point for code point.

Comparing two strings in an **ASCII case-insensitive** manner means comparing them exactly, code point for code point, except that the characters in the range U+0041 to U+005A (i.e., LATIN CAPITAL LETTER A to LATIN CAPITAL LETTER Z) and the corresponding characters in the range U+0061 to U+007A (i.e., LATIN SMALL LETTER A to LATIN SMALL LETTER Z) are considered to also match.

Comparing two strings in a **compatibility caseless** manner means using the Unicode *compatibility caseless match* operation to compare the two strings, with no language-specific tailorings. [\[UNICODE\]](#)

Except where otherwise stated, string comparisons must be performed in a [case-sensitive](#) manner.

Converting a string to ASCII uppercase means replacing all characters in the range U+0061 to U+007A (i.e., LATIN SMALL LETTER A to LATIN SMALL LETTER Z) with the corresponding characters in the range U+0041 to U+005A (i.e., LATIN CAPITAL LETTER A to LATIN CAPITAL LETTER Z).

Converting a string to ASCII lowercase means replacing all characters in the range U+0041 to U+005A (i.e., LATIN CAPITAL LETTER A to LATIN CAPITAL LETTER Z) with the corresponding characters in the range U+0061 to U+007A (i.e., LATIN SMALL LETTER A to LATIN SMALL LETTER Z).

A string *pattern* is a **prefix match** for a string *s* when *pattern* is not longer than *s* and truncating *s* to *pattern*'s length leaves the two strings as matches of each other.

§ 2.4. Common microsyntaxes

There are various places in HTML that accept particular data types, such as dates or numbers. This section describes what the conformance criteria for content in those formats is, and how to parse them.

NOTE:

Implementors are strongly urged to carefully examine any third-party libraries they might consider using to implement the parsing of syntaxes described below. For example, date libraries are likely to implement error handling behavior that differs from what is required in this specification, since error-handling behavior is often not defined in specifications that describe date syntaxes similar to those used in this specification, and thus implementations tend to vary greatly in how they handle errors.

§ 2.4.1. Common parser idioms

The **space characters**, for the purposes of this specification, are U+0020 SPACE, U+0009 CHARACTER TABULATION (tab), U+000A LINE FEED (LF), U+000C FORM FEED (FF), and U+000D CARRIAGE RETURN (CR).

The **White_Space characters** are those that have the Unicode property "White_Space" in the Unicode PropList.txt data file. [\[UNICODE\]](#)

NOTE:

This should not be confused with the "White_Space" value (abbreviated "WS") of the "Bidi_Class" property in the Unicode.txt data file.

The **control characters** are those whose Unicode "General_Category" property has the value "Cc" in the Unicode

UnicodeData.txt data file. [\[UNICODE\]](#)

The **uppercase ASCII letters** are the characters in the range U+0041 LATIN CAPITAL LETTER A to U+005A LATIN CAPITAL LETTER Z.

The **lowercase ASCII letters** are the characters in the range U+0061 LATIN SMALL LETTER A to U+007A LATIN SMALL LETTER Z.

The **ASCII digits** are the characters in the range U+0030 DIGIT ZERO (0) to U+0039 DIGIT NINE (9).

The **alphanumeric ASCII characters** are those that are either [uppercase ASCII letters](#), [lowercase ASCII letters](#), or [ASCII digits](#).

The **ASCII hex digits** are the characters in the ranges U+0030 DIGIT ZERO (0) to U+0039 DIGIT NINE (9), U+0041 LATIN CAPITAL LETTER A to U+0046 LATIN CAPITAL LETTER F, and U+0061 LATIN SMALL LETTER A to U+0066 LATIN SMALL LETTER F.

The **uppercase ASCII hex digits** are the characters in the ranges U+0030 DIGIT ZERO (0) to U+0039 DIGIT NINE (9) and U+0041 LATIN CAPITAL LETTER A to U+0046 LATIN CAPITAL LETTER F only.

The **lowercase ASCII hex digits** are the characters in the ranges U+0030 DIGIT ZERO (0) to U+0039 DIGIT NINE (9) and U+0061 LATIN SMALL LETTER A to U+0066 LATIN SMALL LETTER F only.

Some of the micro-parsers described below follow the pattern of having an *input* variable that holds the string being parsed, and having a *position* variable pointing at the next character to parse in *input*.

For parsers based on this pattern, a step that requires the user agent to **collect a sequence of characters** means that the following algorithm must be run, with *characters* being the set of characters that can be collected:

1. Let *input* and *position* be the same variables as those of the same name in the algorithm that invoked these steps.
2. Let *result* be the empty string.
3. While *position* doesn't point past the end of *input* and the character at *position* is one of the *characters*, append that character to the end of *result* and advance *position* to the next character in *input*.
4. Return *result*.

The step **skip whitespace** means that the user agent must [collect a sequence of characters](#) that are [space characters](#). The collected characters are not used.

When a user agent is to **strip line breaks** from a string, the user agent must remove any U+000A LINE FEED (LF) and U+000D CARRIAGE RETURN (CR) characters from that string.

When a user agent is to **strip leading and trailing whitespace** from a string, the user agent must remove all [space characters](#) that are at the start or end of the string.

When a user agent is to **strip and collapse whitespace** in a string, it must replace any sequence of one or more consecutive [space characters](#) in that string with a single U+0020 SPACE character, and then [strip leading and trailing whitespace](#) from that string.

When a user agent has to **strictly split a string** on a particular delimiter character *delimiter*, it must use the following algorithm:

1. Let *input* be the string being parsed.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
3. Let *tokens* be an ordered list of tokens, initially empty.
4. While *position* is not past the end of *input*:
 1. [Collect a sequence of characters](#) that are not the *delimiter* character.
 2. Append the string collected in the previous step to *tokens*.

3. Advance *position* to the next character in *input*.

5. Return *tokens*.

NOTE:

For the special cases of splitting a string [on spaces](#) and [on commas](#), this algorithm does not apply (those algorithms also perform [whitespace trimming](#)).

§ 2.4.2. Boolean attributes

A number of attributes are **boolean attributes**. The presence of a boolean attribute on an element represents the true value, and the absence of the attribute represents the false value.

If the attribute is present, its value must either be the empty string or a value that is an [ASCII case-insensitive](#) match for the attribute's canonical name, with no leading or trailing whitespace.

NOTE:

The values "true" and "false" are not allowed on [boolean attributes](#). To represent a false value, the attribute has to be omitted altogether.

EXAMPLE 45

Here is an example of a checkbox that is checked and disabled. The checked and disabled attributes are the [boolean attributes](#).

```
<label><input type=checkbox checked name=cheese disabled> Cheese</label>
```

This could be equivalently written as this:

```
<label><input type=checkbox checked=checked name=cheese disabled=disabled> Cheese</label>
```

You can also mix styles; the following is still equivalent:

```
<label><input type='checkbox' checked name=cheese disabled=""> Cheese</label>
```

§ 2.4.3. Keywords and enumerated attributes

Some attributes are defined as taking one of a finite set of keywords. Such attributes are called **enumerated attributes**. The keywords are each defined to map to a particular *state* (several keywords might map to the same state, in which case some of the keywords are synonyms of each other; additionally, some of the keywords can be said to be non-conforming, and are only in the specification for historical reasons). In addition, two default states can be given. The first is the *invalid value default*, the second is the *missing value default*.

If an enumerated attribute is specified, the attribute's value must be an [ASCII case-insensitive](#) match for one of the given keywords that are not said to be non-conforming, with no leading or trailing whitespace.

When the attribute is specified, if its value is an [ASCII case-insensitive](#) match for one of the given keywords then that keyword's state is the state that the attribute represents. If the attribute value matches none of the given keywords, but the attribute has an *invalid value default*, then the attribute represents that state. Otherwise, if the attribute value matches none of the keywords but there is a *missing value default* state defined, then *that* is the state represented by the attribute. Otherwise, there is no default, and invalid values mean that there is no state represented.

When the attribute is *not* specified, if there is a *missing value default* state defined, then that is the state represented by the (missing) attribute. Otherwise, the absence of the attribute means that there is no state represented.

NOTE:

The empty string can be a valid keyword.

§ 2.4.4. Numbers

§ 2.4.4.1. Signed integers

A string is a **valid integer** if it consists of one or more [ASCII digits](#), optionally prefixed with a U+002D HYPHEN-MINUS character (-).

A [valid integer](#) without a U+002D HYPHEN-MINUS (-) prefix represents the number that is represented in base ten by that string of digits. A [valid integer](#) with a U+002D HYPHEN-MINUS (-) prefix represents the number represented in base ten by the string of digits that follows the U+002D HYPHEN-MINUS, subtracted from zero.

The **rules for parsing integers** are as given in the following algorithm. When invoked, the steps must be followed in the order given, aborting at the first step that returns a value. This algorithm will return either an integer or an error.

1. Let `input` be the string being parsed.
2. Let `position` be a pointer into `input`, initially pointing at the start of the string.
3. Let `sign` have the value "positive".
4. [Skip whitespace](#).
5. If `position` is past the end of `input`, return an error.
6. If the character indicated by `position` (the first character) is a U+002D HYPHEN-MINUS character (-):
 1. Let `sign` be "negative".
 2. Advance `position` to the next character.
 3. If `position` is past the end of `input`, return an error.
- Otherwise, if the character indicated by `position` (the first character) is a U+002B PLUS SIGN character (+):
 1. Advance `position` to the next character. (The "+" is ignored, but it is not conforming.)
 2. If `position` is past the end of `input`, return an error.
7. If the character indicated by `position` is not an [ASCII digit](#), then return an error.
8. [Collect a sequence of characters](#) that are [ASCII digits](#), and interpret the resulting sequence as a base-ten integer. Let `value` be that integer.
9. If `sign` is "positive", return `value`, otherwise return the result of subtracting `value` from zero.

§ 2.4.4.2. Non-negative integers

A string is a **valid non-negative integer** if it consists of one or more [ASCII digits](#).

A [valid non-negative integer](#) represents the number that is represented in base ten by that string of digits.

The **rules for parsing non-negative integers** are as given in the following algorithm. When invoked, the steps must be followed in the order given, aborting at the first step that returns a value. This algorithm will return either zero, a positive integer, or an error.

1. Let `input` be the string being parsed.
2. Let `value` be the result of parsing `input` using the [rules for parsing integers](#).
3. If `value` is an error, return an error.
4. If `value` is less than zero, return an error.
5. Return `value`.

§ 2.4.4.3. Floating-point numbers

A string is a **valid floating-point number** if it consists of:

1. Optionally, a U+002D HYPHEN-MINUS character (-).
2. One or both of the following, in the given order:
 1. A series of one or more [ASCII digits](#).
 2. Both of the following, in the given order:
 1. A single U+002E FULL STOP character (.
 2. A series of one or more [ASCII digits](#).
3. Optionally:
 1. Either a U+0065 LATIN SMALL LETTER E character (e) or a U+0045 LATIN CAPITAL LETTER E character (E).
 2. Optionally, a U+002D HYPHEN-MINUS character (-) or U+002B PLUS SIGN character (+).
 3. A series of one or more [ASCII digits](#).

A [valid floating-point number](#) represents the number obtained by multiplying the significand by ten raised to the power of the exponent, where the significand is the first number, interpreted as base ten (including the decimal point and the number after the decimal point, if any, and interpreting the significand as a negative number if the whole string starts with a U+002D HYPHEN-MINUS character (-) and the number is not zero), and where the exponent is the number after the E, if any (interpreted as a negative number if there is a U+002D HYPHEN-MINUS character (-) between the E and the number and the number is not zero, or else ignoring a U+002B PLUS SIGN character (+) between the E and the number if there is one). If there is no E, then the exponent is treated as zero.

NOTE:

The Infinity and Not-a-Number (NaN) values are not [valid floating-point numbers](#).

The **best representation of the number *n* as a floating-point number** is the string obtained from running `ToString(n)`. The abstract operation `ToString` is not uniquely determined. When there are multiple possible strings that could be obtained from `ToString` for a particular value, the user agent must always return the same string for that value (though it may differ from the value used by other user agents).

The **rules for parsing floating-point number values** are as given in the following algorithm. This algorithm must be aborted at the first step that returns something. This algorithm will return either a number or an error.

1. Let *input* be the string being parsed.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
3. Let *value* have the value 1.
4. Let *divisor* have the value 1.
5. Let *exponent* have the value 1.
6. [Skip whitespace](#).
7. If *position* is past the end of *input*, return an error.
8. If the character indicated by *position* is a U+002D HYPHEN-MINUS character (-):
 1. Change *value* and *divisor* to -1.
 2. Advance *position* to the next character.
 3. If *position* is past the end of *input*, return an error.

Otherwise, if the character indicated by *position* (the first character) is a U+002B PLUS SIGN character (+):

1. Advance *position* to the next character. (The "+" is ignored, but it is not conforming.)
 2. If *position* is past the end of *input*, return an error.
9. If the character indicated by *position* is a U+002E FULL STOP (.), and that is not the last character in *input*, and the character after the character indicated by *position* is an [ASCII digit](#), then set *value* to zero and jump to the step labeled *fraction*.
10. If the character indicated by *position* is not an [ASCII digit](#), then return an error.
11. [Collect a sequence of characters](#) that are [ASCII digits](#), and interpret the resulting sequence as a base-ten integer. Multiply *value* by that integer.
12. If *position* is past the end of *input*, jump to the step labeled *conversion*.
13. *Fraction*: If the character indicated by *position* is a U+002E FULL STOP (.), run these substeps:
1. Advance *position* to the next character.
 2. If *position* is past the end of *input*, or if the character indicated by *position* is not an [ASCII digit](#), U+0065 LATIN SMALL LETTER E (e), or U+0045 LATIN CAPITAL LETTER E (E), then jump to the step labeled *conversion*.
 3. If the character indicated by *position* is a U+0065 LATIN SMALL LETTER E character (e) or a U+0045 LATIN CAPITAL LETTER E character (E), skip the remainder of these substeps.
 4. *Fraction loop*: Multiply *divisor* by ten.
 5. Add the value of the character indicated by *position*, interpreted as a base-ten digit (0..9) and divided by *divisor*, to *value*.
 6. Advance *position* to the next character.
 7. If *position* is past the end of *input*, then jump to the step labeled *conversion*.
 8. If the character indicated by *position* is an [ASCII digit](#), jump back to the step labeled *fraction loop* in these substeps.
14. If the character indicated by *position* is a U+0065 LATIN SMALL LETTER E character (e) or a U+0045 LATIN CAPITAL LETTER E character (E), run these substeps:
1. Advance *position* to the next character.
 2. If *position* is past the end of *input*, then jump to the step labeled *conversion*.
 3. If the character indicated by *position* is a U+002D HYPHEN-MINUS character (-):

1. Change *exponent* to -1.
2. Advance *position* to the next character.
3. If *position* is past the end of *input*, then jump to the step labeled *conversion*.

Otherwise, if the character indicated by *position* is a U+002B PLUS SIGN character (+):

1. Advance *position* to the next character.
2. If *position* is past the end of *input*, then jump to the step labeled *conversion*.
3. If the character indicated by *position* is not an [ASCII digit](#), then jump to the step labeled *conversion*.
4. [Collect a sequence of characters](#) that are [ASCII digits](#), and interpret the resulting sequence as a base-ten integer. Multiply *exponent* by that integer.
5. Multiply *value* by ten raised to the *exponent*th power.

15. *Conversion*: Let S be the set of finite IEEE 754 double-precision floating-point values except -0, but with two special values added: 2^{1024} and -2^{1024} .
16. Let *rounded-value* be the number in S that is closest to *value*, selecting the number with an even significand if there are two equally close values. (The two special values 2^{1024} and -2^{1024} are considered to have even significands for this purpose.)
17. If *rounded-value* is 2^{1024} or -2^{1024} , return an error.
18. Return *rounded-value*.

§ 2.4.4.4. Percentages and lengths

The **rules for parsing dimension values** are as given in the following algorithm. When invoked, the steps must be followed in the order given, aborting at the first step that returns a value. This algorithm will return either a number greater than or equal to 0.0, or an error; if a number is returned, then it is further categorized as either a percentage or a length.

1. Let *input* be the string being parsed.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
3. [Skip whitespace](#).
4. If *position* is past the end of *input*, return an error.
5. If the character indicated by *position* is a U+002B PLUS SIGN character (+), advance *position* to the next character.
6. If *position* is past the end of *input*, return an error.
7. If the character indicated by *position* is not an [ASCII digit](#), then return an error.
8. [Collect a sequence of characters](#) that are [ASCII digits](#), and interpret the resulting sequence as a base-ten integer. Let *value* be that number.
9. If *position* is past the end of *input*, return *value* as a length.
10. If the character indicated by *position* is a U+002E FULL STOP character (.):
 1. Advance *position* to the next character.
 2. If *position* is past the end of *input*, or if the character indicated by *position* is not an [ASCII digit](#), then return *value* as a length.
 3. Let *divisor* have the value 1.
 4. *Fraction loop*: Multiply *divisor* by ten.
 5. Add the value of the character indicated by *position*, interpreted as a base-ten digit (0..9) and divided by *divisor*, to *value*.
 6. Advance *position* to the next character.
 7. If *position* is past the end of *input*, then return *value* as a length.
 8. If the character indicated by *position* is an [ASCII digit](#), return to the step labeled *fraction loop* in these substeps.
11. If *position* is past the end of *input*, return *value* as a length.
12. If the character indicated by *position* is a U+0025 PERCENT SIGN character (%), return *value* as a percentage.
13. Return *value* as a length.

§ 2.4.4.5. Non-zero percentages and lengths

The **rules for parsing non-zero dimension values** are as given in the following algorithm. When invoked, the steps must

be followed in the order given, aborting at the first step that returns a value. This algorithm will return either a number greater than 0.0, or an error; if a number is returned, then it is further categorized as either a percentage or a length.

1. Let *input* be the string being parsed.
2. Let *value* be the result of parsing *input* using the [rules for parsing dimension values](#).
3. If *value* is an error, return an error.
4. If *value* is zero, return an error.
5. If *value* is a percentage, return *value* as a percentage.
6. Return *value* as a length.

§ 2.4.4.6. Lists of floating-point numbers

A **valid list of floating-point numbers** is a number of [valid floating-point numbers](#) separated by U+002C COMMA characters, with no other characters (e.g. no [space characters](#)). In addition, there might be restrictions on the number of floating-point numbers that can be given, or on the range of values allowed.

The **rules for parsing a list of floating-point numbers** are as follows:

1. Let *input* be the string being parsed.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
3. Let *numbers* be an initially empty list of floating-point numbers. This list will be the result of this algorithm.
4. [Collect a sequence of characters](#) that are [space characters](#), U+002C COMMA, or U+003B SEMICOLON characters. This skips past any leading delimiters.
5. While *position* is not past the end of *input*:
 1. [Collect a sequence of characters](#) that are not [space characters](#), U+002C COMMA, U+003B SEMICOLON, [ASCII digits](#), U+002E FULL STOP, or U+002D HYPHEN-MINUS characters. This skips past leading garbage.
 2. [Collect a sequence of characters](#) that are not [space characters](#), U+002C COMMA, or U+003B SEMICOLON characters, and let *unparsed number* be the result.
 3. Let *number* be the result of parsing *unparsed number* using the [rules for parsing floating-point number values](#).
 4. If *number* is an error, set *number* to zero.
 5. Append *number* to *numbers*.
 6. [Collect a sequence of characters](#) that are [space characters](#), U+002C COMMA, or U+003B SEMICOLON characters. This skips past the delimiter.
6. Return *numbers*.

§ 2.4.4.7. Lists of dimensions

The **rules for parsing a list of dimensions** are as follows. These rules return a list of zero or more pairs consisting of a number and a unit, the unit being one of *percentage*, *relative*, and *absolute*.

1. Let raw input be the string being parsed.
2. If the last character in *raw input* is a U+002C COMMA character (,), then remove that character from *raw input*.
3. [Split the string *raw input* on commas](#). Let *raw tokens* be the resulting list of tokens.
4. Let *result* be an empty list of number/unit pairs.
5. For each token in *raw tokens*, run the following substeps:

1. Let *input* be the token.
 2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
 3. Let *value* be the number 0.
 4. Let *unit* be *absolute*.
 5. If *position* is past the end of *input*, set *unit* to *relative* and jump to the last substep.
 6. If the character at *position* is an [ASCII digit](#), [collect a sequence of characters](#) that are [ASCII digits](#), interpret the resulting sequence as an integer in base ten, and increment *value* by that integer.
 7. If the character at *position* is a U+002E FULL STOP character (.), run these substeps:
 1. [Collect a sequence of characters](#) consisting of [space characters](#) and [ASCII digits](#). Let *s* be the resulting sequence.
 2. Remove all [space characters](#) in *s*.
 3. If *s* is not the empty string, run these subsubsteps:
 1. Let *length* be the number of characters in *s* (after the spaces were removed).
 2. Let *fraction* be the result of interpreting *s* as a base-ten integer, and then dividing that number by 10^{length} .
 3. Increment *value* by *fraction*.
 8. [Skip whitespace](#).
 9. If the character at *position* is a U+0025 PERCENT SIGN character (%), then set *unit* to *percentage*.
 Otherwise, if the character at *position* is a U+002A ASTERISK character (*), then set *unit* to *relative*.
 10. Add an entry to *result* consisting of the number given by *value* and the unit given by *unit*.
6. Return the list *result*.

§ 2.4.5. Dates and times

This specification encodes dates and times according to a common subset of the [\[ISO8601\]](#) standard for dates.

This means that encoded dates will look like 1582-03-01, 0033-03-27, or 2016-03-01, and date-times will look like 1929-11-13T19:00Z, 0325-06-03T00:21+10:30. The format is approximately YYYY-MM-DDTHH:MM:SS.DD±HH:MM, although some parts are optional, for example to express a month and day as in a birthday, a time without time-zone information, and the like.

Times are expressed using the 24-hour clock, and it is as error to express leap seconds.

Dates are expressed in the [proleptic Gregorian calendar](#) between the proleptic year 0000, and the year 9999. Other years cannot be encoded.

The [proleptic Gregorian calendar](#) is the calendar most common globally since around 1950, and is likely to be understood by almost everyone for dates between the years 1950 and 9999, and for many people for dates in the last few decades or centuries.

The Gregorian calendar was adopted officially in different countries at different times, between the years 1582 when it was proposed by Pope Gregory XIII as a replacement for the Julian calendar, and 1949 when it was adopted by the People's republic of China.

For most practical purposes, dealing with the present, recent past, or the next few thousand years, this will work without problems. For dates before the adoption of the Gregorian Calendar - for example prior to 1917 in Russia or Turkey, prior to 1752 in Britain or the then British colonies of America, or prior to 1582 in Spain, the Spanish colonies in America, and the rest of the world, dates will not match those written at the time.

The use of the Gregorian calendar as an underlying encoding is a somewhat arbitrary choice. Many other calendars were or are in use, and the interested reader should look for information on the Web.

See also the discussion of [date, time, and number formats](#) in forms (for authors), [implementation notes regarding localization of form controls](#), and the `<time>` element.

In the algorithms below, the **number of days in month** `month` of year `year` is: 31 if `month` is 1, 3, 5, 7, 8, 10, or 12; 30 if `month` is 4, 6, 9, or 11; 29 if `month` is 2 and `year` is a number divisible by 400, or if `year` is a number divisible by 4 but not by 100; and 28 otherwise. This takes into account leap years in the Gregorian calendar. [\[GREGORIAN\]](#)

When [ASCII digits](#) are used in the date and time syntaxes defined in this section, they express numbers in base ten.

NOTE:

While the formats described here are intended to be subsets of the corresponding ISO8601 formats, this specification defines parsing rules in much more detail than ISO8601. Implementors are therefore encouraged to carefully examine any date parsing libraries before using them to implement the parsing rules described below; ISO8601 libraries might not parse dates and times in exactly the same manner. [\[ISO8601\]](#)

Where this specification refers to the **proleptic Gregorian calendar**, it means the modern Gregorian calendar, extrapolated backwards to year 1. A date in the [proleptic Gregorian calendar](#), sometimes explicitly referred to as a **proleptic-Gregorian date**, is one that is described using that calendar even if that calendar was not in use at the time (or place) in question. [\[GREGORIAN\]](#)

2.4.5.1. Months

A **month** consists of a specific [proleptic-Gregorian date](#) with no time-zone information and no date information beyond a year and a month. [\[GREGORIAN\]](#)

A string is a **valid month string** representing a year `year` and month `month` if it consists of the following components in the given order:

1. Four or more [ASCII digits](#), representing `year`, where `year` > 0
2. A U+002D HYPHEN-MINUS character (-)
3. Two [ASCII digits](#), representing the month `month`, in the range $1 \leq \text{month} \leq 12$

EXAMPLE 46

For example, February 2005 is encoded 2005-02, and March of the year 33AD (as a proleptic gregorian date) is encoded 0033-03. The expression 325-03 does *not* mean March in the year 325, it is an error, because it does not have 4 digits for the year.

The rules to **parse a month string** are as follows. This will return either a year and month, or nothing. If at any point the algorithm says that it "fails", this means that it is aborted at that point and returns nothing.

1. Let *input* be the string being parsed.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
3. [Parse a month component](#) to obtain *year* and *month*. If this returns nothing, then fail.
4. If *position* is *not* beyond the end of *input*, then fail.
5. Return *year* and *month*.

The rules to **parse a month component**, given an *input* string and a *position*, are as follows. This will return either a year and a month, or nothing. If at any point the algorithm says that it "fails", this means that it is aborted at that point and returns nothing.

1. [Collect a sequence of characters](#) that are [ASCII digits](#). If the collected sequence is not at least four characters long, then fail. Otherwise, interpret the resulting sequence as a base-ten integer. Let that number be the *year*.
2. If *year* is not a number greater than zero, then fail.
3. If *position* is beyond the end of *input* or if the character at *position* is not a U+002D HYPHEN-MINUS character, then fail. Otherwise, move *position* forwards one character.
4. [Collect a sequence of characters](#) that are [ASCII digits](#). If the collected sequence is not exactly two characters long, then fail. Otherwise, interpret the resulting sequence as a base-ten integer. Let that number be the *month*.
5. If *month* is not a number in the range $1 \leq \textit{month} \leq 12$, then fail.
6. Return *year* and *month*.

2.4.5.2. Dates

A **date** consists of a specific [proleptic-Gregorian date](#) with no time-zone information, consisting of a year, a month, and a day. [\[GREGORIAN\]](#)

A string is a **valid date string** representing a year *year*, month *month*, and day *day* if it consists of the following components in the given order:

1. A [valid month string](#), representing *year* and *month*
2. A U+002D HYPHEN-MINUS character (-)
3. Two [ASCII digits](#), representing *day*, in the range $1 \leq \textit{day} \leq \textit{maxday}$ where *maxday* is the [number of days in the month *month* and year *year*](#).

EXAMPLE 47

For example, 29 February 2016 is encoded 2016-02-29, and 3 March of the year 33AD (as a proleptic gregorian date) is encoded 0033-03-03. The expression 325-03-03 does *not* mean 3 March in the year 325, it is an error, because it does not have 4 digits for the year.

The rules to **parse a date string** are as follows. This will return either a date, or nothing. If at any point the algorithm says that it "fails", this means that it is aborted at that point and returns nothing.

1. Let *input* be the string being parsed.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.

3. [Parse a date component](#) to obtain `year`, `month`, and `day`. If this returns nothing, then fail.
4. If `position` is not beyond the end of `input`, then fail.
5. Let `date` be the date with year `year`, month `month`, and day `day`.
6. Return `date`.

The rules to **parse a date component**, given an `input` string and a `position`, are as follows. This will return either a year, a month, and a day, or nothing. If at any point the algorithm says that it "fails", this means that it is aborted at that point and returns nothing.

1. [Parse a month component](#) to obtain `year` and `month`. If this returns nothing, then fail.
2. Let `maxday` be the [number of days in month](#) `month` of year `year`.
3. If `position` is beyond the end of `input` or if the character at `position` is not a U+002D HYPHEN-MINUS character, then fail. Otherwise, move `position` forwards one character.
4. [Collect a sequence of characters](#) that are [ASCII digits](#). If the collected sequence is not exactly two characters long, then fail. Otherwise, interpret the resulting sequence as a base-ten integer. Let that number be the `day`.
5. If `day` is not a number in the range $1 \leq \text{day} \leq \text{maxday}$, then fail.
6. Return `year`, `month`, and `day`.

2.4.5.3. Yearless dates

A **yearless date** consists of a Gregorian month and a day within that month, but with no associated year. [\[GREGORIAN\]](#)

A string is a **valid yearless date string** representing a month `month` and a day `day` if it consists of the following components in the given order:

1. Optionally, two U+002D HYPHEN-MINUS characters (-)
2. Two [ASCII digits](#), representing the month `month`, in the range $1 \leq \text{month} \leq 12$
3. A U+002D HYPHEN-MINUS character (-)
4. Two [ASCII digits](#), representing `day`, in the range $1 \leq \text{day} \leq \text{maxday}$ where `maxday` is the [number of days](#) in the month `month` and any arbitrary leap year (e.g., 4 or 2000)

NOTE:

In other words, if the `month` is "02", meaning February, then the day can be 29, as if the year was a leap year.

EXAMPLE 48

For example, 29 February is encoded 02-29, and 3 March is encoded 03-03.

The rules to **parse a yearless date string** are as follows. This will return either a month and a day, or nothing. If at any point the algorithm says that it "fails", this means that it is aborted at that point and returns nothing.

1. Let `input` be the string being parsed.
2. Let `position` be a pointer into `input`, initially pointing at the start of the string.
3. [Parse a yearless date component](#) to obtain `month` and `day`. If this returns nothing, then fail.
4. If `position` is not beyond the end of `input`, then fail.
5. Return `month` and `day`.

The rules to **parse a yearless date component**, given an `input` string and a `position`, are as follows. This will return either a month and a day, or nothing. If at any point the algorithm says that it "fails", this means that it is aborted at that point and returns nothing.

1. [Collect a sequence of characters](#) that are U+002D HYPHEN-MINUS characters (-). If the collected sequence is not exactly zero or two characters long, then fail.
2. [Collect a sequence of characters](#) that are [ASCII digits](#). If the collected sequence is not exactly two characters long, then fail. Otherwise, interpret the resulting sequence as a base-ten integer. Let that number be the *month*.
3. If *month* is not a number in the range $1 \leq \text{month} \leq 12$, then fail.
4. Let *maxday* be the [number of days](#) in month *month* of any arbitrary leap year (e.g., 4 or 2000).
5. If *position* is beyond the end of *input* or if the character at *position* is not a U+002D HYPHEN-MINUS character, then fail. Otherwise, move *position* forwards one character.
6. [Collect a sequence of characters](#) that are [ASCII digits](#). If the collected sequence is not exactly two characters long, then fail. Otherwise, interpret the resulting sequence as a base-ten integer. Let that number be the *day*.
7. If *day* is not a number in the range $1 \leq \text{day} \leq \text{maxday}$, then fail.
8. Return *month* and *day*.

§ 2.4.5.4. Times

A **time** consists of a specific time with no time-zone information, consisting of an hour, a minute, a second, and a fraction of a second.

A string is a **valid time string** representing an hour *hour*, a minute *minute*, and a second *second* if it consists of the following components in the given order:

1. Two [ASCII digits](#), representing *hour*, in the range $0 \leq \text{hour} \leq 23$
2. A U+003A COLON character (:))
3. Two [ASCII digits](#), representing *minute*, in the range $0 \leq \text{minute} \leq 59$
4. If *second* is non-zero, or optionally if *second* is zero:
 1. A U+003A COLON character (:))
 2. Two [ASCII digits](#), representing the integer part of *second*, in the range $0 \leq s \leq 59$
 3. If *second* is not an integer, or optionally if *second* is an integer:
 1. A 002E FULL STOP character (.))
 2. One, two, or three [ASCII digits](#), representing the fractional part of *second*

NOTE:

The *second* component cannot be 60 or 61; leap seconds cannot be represented.

EXAMPLE 49

Times are encoded using the 24 hour clock, with optional seconds, and optional decimal fractions of seconds. Thus 7.45pm is encoded as 19:45. Note that parsing that time will return 19:45:00, or 7.45pm and zero seconds. 19:45:45.456 is 456 thousandths of a second after 7.45pm and 45 seconds.

The rules to **parse a time string** are as follows. This will return either a time, or nothing. If at any point the algorithm says that it "fails", this means that it is aborted at that point and returns nothing.

1. Let *input* be the string being parsed.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
3. [Parse a time component](#) to obtain *hour*, *minute*, and *second*. If this returns nothing, then fail.
4. If *position* is not beyond the end of *input*, then fail.

5. Let *time* be the time with hour *hour*, minute *minute*, and second *second*.

6. Return *time*.

The rules to **parse a time component**, given an *input* string and a *position*, are as follows. This will return either an hour, a minute, and a second, or nothing. If at any point the algorithm says that it "fails", this means that it is aborted at that point and returns nothing.

1. [Collect a sequence of characters](#) that are [ASCII digits](#). If the collected sequence is not exactly two characters long, then fail. Otherwise, interpret the resulting sequence as a base-ten integer. Let that number be the *hour*.
2. If *hour* is not a number in the range $0 \leq \text{hour} \leq 23$, then fail.
3. If *position* is beyond the end of *input* or if the character at *position* is not a U+003A COLON character, then fail. Otherwise, move *position* forwards one character.
4. [Collect a sequence of characters](#) that are [ASCII digits](#). If the collected sequence is not exactly two characters long, then fail. Otherwise, interpret the resulting sequence as a base-ten integer. Let that number be the *minute*.
5. If *minute* is not a number in the range $0 \leq \text{minute} \leq 59$, then fail.
6. Let *second* be a string with the value "0".
7. If *position* is not beyond the end of *input* and the character at *position* is a U+003A COLON, then run these substeps:
 1. Advance *position* to the next character in *input*.
 2. If *position* is beyond the end of *input*, or at the last character in *input*, or if the next *two* characters in *input* starting at *position* are not both [ASCII digits](#), then fail.
 3. [Collect a sequence of characters](#) that are either [ASCII digits](#) or U+002E FULL STOP characters. If the collected sequence is three characters long, or if it is longer than three characters long and the third character is not a U+002E FULL STOP character, or if it has more than one U+002E FULL STOP character, then fail. Otherwise, let *second* be the collected string.
8. Interpret *second* as a base-ten number (possibly with a fractional part). Let *second* be that number instead of the string version.
9. If *second* is not a number in the range $0 \leq \text{second} < 60$, then fail.
10. Return *hour*, *minute*, and *second*.

2.4.5.5. Floating dates and times

A **floating date and time** consists of a specific [proleptic-Gregorian date](#), consisting of a year, a month, and a day, and a time, consisting of an hour, a minute, a second, and a fraction of a second, but expressed without a time zone. [\[GREGORIAN\]](#)

A string is a **valid floating date and time string** representing a date and time if it consists of the following components in the given order:

1. A [valid date string](#) representing the date
2. A U+0054 LATIN CAPITAL LETTER T character (T) or a U+0020 SPACE character
3. A [valid time string](#) representing the time

A string is a **valid normalized floating date and time string** representing a date and time if it consists of the following components in the given order:

1. A [valid date string](#) representing the date
2. A U+0054 LATIN CAPITAL LETTER T character (T)
3. A [valid time string](#) representing the time, expressed as the shortest possible string for the given time (e.g., omitting the

seconds component entirely if the given time is zero seconds past the minute)

The rules to **parse a floating date and time string** are as follows. This will return either a date and time, or nothing. If at any point the algorithm says that it "fails", this means that it is aborted at that point and returns nothing.

1. Let *input* be the string being parsed.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
3. [Parse a date component](#) to obtain *year*, *month*, and *day*. If this returns nothing, then fail.
4. If *position* is beyond the end of *input* or if the character at *position* is neither a U+0054 LATIN CAPITAL LETTER T character (T) nor a U+0020 SPACE character, then fail. Otherwise, move *position* forwards one character.
5. [Parse a time component](#) to obtain *hour*, *minute*, and *second*. If this returns nothing, then fail.
6. If *position* is not beyond the end of *input*, then fail.
7. Let *date* be the date with year *year*, month *month*, and day *day*.
8. Let *time* be the time with hour *hour*, minute *minute*, and second *second*.
9. Return *date* and *time*.

§ 2.4.5.6. Time zones

A **time-zone offset** consists of a signed number of hours and minutes.

A string is a **valid time-zone offset string** representing a time-zone offset if it consists of either:

- A U+005A LATIN CAPITAL LETTER Z character (Z), allowed only if the time zone is UTC
- Or, the following components, in the given order:
 1. Either a U+002B PLUS SIGN character (+) or, if the time-zone offset is not zero, a U+002D HYPHEN-MINUS character (-), representing the sign of the time-zone offset
 2. Two [ASCII digits](#), representing the hours component *hour* of the time-zone offset, in the range $0 \leq \textit{hour} \leq 23$
 3. Optionally, a U+003A COLON character (:)
 4. Two [ASCII digits](#), representing the minutes component *minute* of the time-zone offset, in the range $0 \leq \textit{minute} \leq 59$

NOTE:

This format allows for time-zone offsets from -23:59 to +23:59. In practice, however, right now the range of offsets of actual time zones is -12:00 to +14:00, and the minutes component of offsets of actual time zones is always either 00, 30, or 45. There is no guarantee that this will remain so forever, however; time zones are changed by countries at will and do not follow a standard.

NOTE:

See also the usage notes and examples in the [global date and time](#) section below for details on using time-zone offsets with historical times that predate the formation of formal time zones.

The rules to **parse a time-zone offset string** are as follows. This will return either a time-zone offset, or nothing. If at any point the algorithm says that it "fails", this means that it is aborted at that point and returns nothing.

1. Let *input* be the string being parsed.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
3. [Parse a time-zone offset component](#) to obtain *timezone_{hours}* and *timezone_{minutes}*. If this returns nothing, then fail.
4. If *position* is not beyond the end of *input*, then fail.

5. Return the time-zone offset that is $timezone_{hours}$ hours and $timezone_{minutes}$ minutes from UTC.

The rules to **parse a time-zone offset component**, given an $input$ string and a $position$, are as follows. This will return either time-zone hours and time-zone minutes, or nothing. If at any point the algorithm says that it "fails", this means that it is aborted at that point and returns nothing.

1. If the character at $position$ is a U+005A LATIN CAPITAL LETTER Z character (Z), then:

1. Let $timezone_{hours}$ be 0.
2. Let $timezone_{minutes}$ be 0.
3. Advance $position$ to the next character in $input$.

Otherwise, if the character at $position$ is either a U+002B PLUS SIGN (+) or a U+002D HYPHEN-MINUS (-), then:

1. If the character at $position$ is a U+002B PLUS SIGN (+), let $sign$ be "positive". Otherwise, it's a U+002D HYPHEN-MINUS (-); let $sign$ be "negative".
2. Advance $position$ to the next character in $input$.
3. [Collect a sequence of characters](#) that are [ASCII digits](#). Let s be the collected sequence.
4. If s is exactly two characters long, then run these substeps:

1. Interpret s as a base-ten integer. Let that number be the $timezone_{hours}$.
2. If $position$ is beyond the end of $input$ or if the character at $position$ is not a U+003A COLON character, then fail. Otherwise, move $position$ forwards one character.
3. [Collect a sequence of characters](#) that are [ASCII digits](#). If the collected sequence is not exactly two characters long, then fail. Otherwise, interpret the resulting sequence as a base-ten integer. Let that number be the $timezone_{minutes}$.

If s is exactly four characters long, then run these substeps:

1. Interpret the first two characters of s as a base-ten integer. Let that number be the $timezone_{hours}$.
2. Interpret the last two characters of s as a base-ten integer. Let that number be the $timezone_{minutes}$.

Otherwise, fail.

5. If $timezone_{hours}$ is not a number in the range $0 \leq timezone_{hours} \leq 23$, then fail.
6. If $sign$ is "negative", then negate $timezone_{hours}$.
7. If $timezone_{minutes}$ is not a number in the range $0 \leq timezone_{minutes} \leq 59$, then fail.
8. If $sign$ is "negative", then negate $timezone_{minutes}$.

Otherwise, fail.

2. Return $timezone_{hours}$ and $timezone_{minutes}$.

§ 2.4.5.7. Global dates and times

A **global date and time** consists of a specific [proleptic-Gregorian date](#), consisting of a year, a month, and a day, and a time, consisting of an hour, a minute, a second, and a fraction of a second, expressed with a time-zone offset, consisting of a signed number of hours and minutes. [\[GREGORIAN\]](#)

A string is a **valid global date and time string** representing a date, time, and a time-zone offset if it consists of the following components in the given order:

1. A [valid date string](#) representing the date
2. A U+0054 LATIN CAPITAL LETTER T character (T) or a U+0020 SPACE character

3. A [valid time string](#) representing the time
4. A [valid time-zone offset string](#) representing the time-zone offset

Times in dates before the formation of UTC in the mid twentieth century must be expressed and interpreted in terms of UT1 (contemporary Earth mean solar time at the 0° longitude), not UTC (the approximation of UT1 that ticks in SI seconds). Time before the formation of time zones must be expressed and interpreted as UT1 times with explicit time zones that approximate the contemporary difference between the appropriate local time and the time observed at the location of Greenwich, London.

EXAMPLE 50

The following are some examples of dates written as [valid global date and time strings](#).

"0037-12-13 00:00Z"

Midnight in areas using London time on the birthday of Nero (the Roman Emperor). See below for further discussion on which date this actually corresponds to.

"1979-10-14T12:00:00.001-04:00"

One millisecond after noon on October 14th 1979, in the time zone in use on the east coast of the USA during daylight saving time.

"8592-01-01T02:09+02:09"

Midnight UTC on the 1st of January, 8592. The time zone associated with that time is two hours and nine minutes ahead of UTC, which is not currently a real time zone, but is nonetheless allowed.

Several things are notable about these dates:

- Years with fewer than four digits have to be zero-padded. The date "37-12-13" would not be a valid date.
- If the "T" is replaced by a space, it must be a single space character. The string "2001-12-21 12:00Z" (with two spaces between the components) would not be parsed successfully.
- To unambiguously identify a moment in time prior to the introduction of the Gregorian calendar (insofar as moments in time before the formation of UTC can be unambiguously identified), the date has to be first converted to the Gregorian calendar from the calendar in use at the time (e.g., from the Julian calendar). The date of Nero's birth is the 15th of December 37, in the Julian Calendar, which is the 13th of December 37 in the [proleptic Gregorian calendar](#).
- The time and time-zone offset components are not optional.
- Dates before the year one can't be represented as a datetime in this version of HTML.
- Times of specific events in ancient times are, at best, approximations, since time was not well coordinated or measured until relatively recent decades.
- Time-zone offsets differ based on daylight savings time.

NOTE:

The zone offset is not a complete time zone specification. When working with real date and time values, consider using a separate field for time zone, perhaps using IANA time zone IDs. [\[TIMEZONE\]](#)

A string is a **valid normalized global date and time string** representing a date, time, and a time-zone offset if it consists of the following components in the given order:

1. A [valid date string](#) representing the date converted to the UTC time zone
2. A U+0054 LATIN CAPITAL LETTER T character (T)
3. A [valid time string](#) representing the time converted to the UTC time zone and expressed as the shortest possible string for the given time (e.g., omitting the seconds component entirely if the given time is zero seconds past the minute)
4. A U+005A LATIN CAPITAL LETTER Z character (Z)

The rules to **parse a global date and time string** are as follows. This will return either a time in UTC, with associated time-zone offset information for round-tripping or display purposes, or nothing. If at any point the algorithm says that it "fails", this means that it is aborted at that point and returns nothing.

1. Let *input* be the string being parsed.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
3. [Parse a date component](#) to obtain *year*, *month*, and *day*. If this returns nothing, then fail.
4. If *position* is beyond the end of *input* or if the character at *position* is neither a U+0054 LATIN CAPITAL LETTER T character (T) nor a U+0020 SPACE character, then fail. Otherwise, move *position* forwards one character.
5. [Parse a time component](#) to obtain *hour*, *minute*, and *second*. If this returns nothing, then fail.
6. If *position* is beyond the end of *input*, then fail.
7. [Parse a time-zone offset component](#) to obtain *timezone_{hours}* and *timezone_{minutes}*. If this returns nothing, then fail.
8. If *position* is not beyond the end of *input*, then fail.
9. Let *time* be the moment in time at year *year*, month *month*, day *day*, hours *hour*, minute *minute*, second *second*, subtracting *timezone_{hours}* hours and *timezone_{minutes}* minutes. That moment in time is a moment in the UTC time zone.
10. Let *timezone* be *timezone_{hours}* hours and *timezone_{minutes}* minutes from UTC.
11. Return *time* and *timezone*.

2.4.5.8. Weeks

A **week** consists of a week-year number and a week number representing a seven-day period starting on a Monday. Each week-year in this calendaring system has either 52 or 53 such seven-day periods, as defined below. The seven-day period starting on the Gregorian date Monday December 29th 1969 (1969-12-29) is defined as week number 1 in week-year 1970. Consecutive weeks are numbered sequentially. The week before the number 1 week in a week-year is the last week in the previous week-year, and vice versa. [\[GREGORIAN\]](#)

A week-year with a number *year* has 53 weeks if it corresponds to either a year *year* in the [proleptic Gregorian calendar](#) that has a Thursday as its first day (January 1st), or a year *year* in the [proleptic Gregorian calendar](#) that has a Wednesday as its first day (January 1st) and where *year* is a number divisible by 400, or a number divisible by 4 but not by 100. All other week-years have 52 weeks.

The **week number of the last day** of a week-year with 53 weeks is 53; the week number of the last day of a week-year with 52 weeks is 52.

NOTE:

The week-year number of a particular day can be different than the number of the year that contains that day in the [proleptic Gregorian calendar](#). The first week in a week-year *y* is the week that contains the first Thursday of the Gregorian year *y*.

NOTE:

For modern purposes, a [week](#) as defined here is equivalent to ISO weeks as defined in ISO 8601. [\[ISO8601\]](#)

A string is a **valid week string** representing a week-year *year* and week *week* if it consists of the following components in the given order:

1. Four or more [ASCII digits](#), representing *year*, where *year* > 0
2. A U+002D HYPHEN-MINUS character (-)
3. A U+0057 LATIN CAPITAL LETTER W character (W)
4. Two [ASCII digits](#), representing the week *week*, in the range $1 \leq \text{week} \leq \text{maxweek}$, where *maxweek* is the [week num-](#)

ber of the last day of week-year *year*.

The rules to **parse a week string** are as follows. This will return either a week-year number and week number, or nothing. If at any point the algorithm says that it "fails", this means that it is aborted at that point and returns nothing.

1. Let *input* be the string being parsed.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
3. **Collect a sequence of characters** that are **ASCII digits**. If the collected sequence is not at least four characters long, then fail. Otherwise, interpret the resulting sequence as a base-ten integer. Let that number be the *year*.
4. If *year* is not a number greater than zero, then fail.
5. If *position* is beyond the end of *input* or if the character at *position* is not a U+002D HYPHEN-MINUS character, then fail. Otherwise, move *position* forwards one character.
6. If *position* is beyond the end of *input* or if the character at *position* is not a U+0057 LATIN CAPITAL LETTER W character (W), then fail. Otherwise, move *position* forwards one character.
7. **Collect a sequence of characters** that are **ASCII digits**. If the collected sequence is not exactly two characters long, then fail. Otherwise, interpret the resulting sequence as a base-ten integer. Let that number be the *week*.
8. Let *maxweek* be the **week number of the last day** of year *year*.
9. If *week* is not a number in the range $1 \leq \textit{week} \leq \textit{maxweek}$, then fail.
10. If *position* is not beyond the end of *input*, then fail.
11. Return the week-year number *year* and the week number *week*.

§ 2.4.5.9. Durations

A **duration** consists of a number of seconds.

NOTE:

Since months and seconds are not comparable (a month is not a precise number of seconds, but is instead a period whose exact length depends on the precise day from which it is measured) a **duration** as defined in this specification cannot include months (or years, which are equivalent to twelve months). Only durations that describe a specific number of seconds can be described.

A string is a **valid duration string** representing a **duration** *t* if it consists of either of the following:

- A literal U+0050 LATIN CAPITAL LETTER P character followed by one or more of the following subcomponents, in the order given, where the number of days, hours, minutes, and seconds corresponds to the same number of seconds as in *t*:
 1. One or more **ASCII digits** followed by a U+0044 LATIN CAPITAL LETTER D character, representing a number of days.
 2. A U+0054 LATIN CAPITAL LETTER T character followed by one or more of the following subcomponents, in the order given:
 1. One or more **ASCII digits** followed by a U+0048 LATIN CAPITAL LETTER H character, representing a number of hours.
 2. One or more **ASCII digits** followed by a U+004D LATIN CAPITAL LETTER M character, representing a number of minutes.
 3. The following components:
 1. One or more **ASCII digits**, representing a number of seconds.
 2. Optionally, a U+002E FULL STOP character (.) followed by one, two, or three **ASCII digits**, represent-

ing a fraction of a second.

3. A U+0053 LATIN CAPITAL LETTER S character.

NOTE:

This, as with a number of other date- and time-related microsyntaxes defined in this specification, is based on one of the formats defined in ISO 8601. [\[ISO8601\]](#)

- One or more [duration time components](#), each with a different [duration time component scale](#), in any order; the sum of the represented seconds being equal to the number of seconds in *t*.

A **duration time component** is a string consisting of the following components:

1. Zero or more [space characters](#).
2. One or more [ASCII digits](#), representing a number of time units, scaled by the [duration time component scale](#) specified (see below) to represent a number of seconds.
3. If the [duration time component scale](#) specified is 1 (i.e., the units are seconds), then, optionally, a U+002E FULL STOP character (.) followed by one, two, or three [ASCII digits](#), representing a fraction of a second.
4. Zero or more [space characters](#).
5. One of the following characters, representing the **duration time component scale** of the time unit used in the numeric part of the [duration time component](#):

U+0057 LATIN CAPITAL LETTER W character

U+0077 LATIN SMALL LETTER W character

Weeks. The scale is 604800.

U+0044 LATIN CAPITAL LETTER D character

U+0064 LATIN SMALL LETTER D character

Days. The scale is 86400.

U+0048 LATIN CAPITAL LETTER H character

U+0068 LATIN SMALL LETTER H character

Hours. The scale is 3600.

U+004D LATIN CAPITAL LETTER M character

U+006D LATIN SMALL LETTER M character

Minutes. The scale is 60.

U+0053 LATIN CAPITAL LETTER S character

U+0073 LATIN SMALL LETTER S character

Seconds. The scale is 1.

6. Zero or more [space characters](#).

NOTE:

This is not based on any of the formats in ISO 8601. It is intended to be a more human-readable alternative to the ISO 8601 duration format.

The rules to **parse a duration string** are as follows. This will return either a [duration](#) or nothing. If at any point the algorithm says that it "fails", this means that it is aborted at that point and returns nothing.

1. Let *input* be the string being parsed.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
3. Let *months*, *seconds*, and *component count* all be zero.
4. Let *M-disambiguator* be *minutes*.

NOTE:

This flag's other value is months. It is used to disambiguate the "M" unit in ISO8601 durations, which use the same unit for months and minutes. Months are not allowed, but are parsed for future compatibility and to avoid misinterpreting ISO8601 durations that would be valid in other contexts.

5. [Skip whitespace.](#)

6. If *position* is past the end of *input*, then fail.

7. If the character in *input* pointed to by *position* is a U+0050 LATIN CAPITAL LETTER P character, then advance *position* to the next character, set *M-disambiguator* to *months*, and [skip whitespace](#).

8. Run the following substeps in a loop, until a step requiring the loop to be broken or the entire algorithm to fail is reached:

1. Let *units* be undefined. It will be assigned one of the following values: *years*, *months*, *weeks*, *days*, *hours*, *minutes*, and *seconds*.

2. Let *next character* be undefined. It is used to process characters from the *input*.

3. If *position* is past the end of *input*, then break the loop.

4. If the character in *input* pointed to by *position* is a U+0054 LATIN CAPITAL LETTER T character, then advance *position* to the next character, set *M-disambiguator* to *minutes*, [skip whitespace](#), and return to the top of the loop.

5. Set *next character* to the character in *input* pointed to by *position*.

6. If *next character* is a U+002E FULL STOP character (.), then let *N* equal zero. (Do not advance *position*. That is taken care of below.)

Otherwise, if *next character* is an [ASCII digit](#), then [collect a sequence of characters](#) that are [ASCII digits](#), interpret the resulting sequence as a base-ten integer, and let *N* be that number.

Otherwise *next character* is not part of a number; fail.

7. If *position* is past the end of *input*, then fail.

8. Set *next character* to the character in *input* pointed to by *position*, and this time advance *position* to the next character. (If *next character* was a U+002E FULL STOP character (.) before, it will still be that character this time.)

9. If *next character* is a U+002E FULL STOP character (.), then run these substeps:

1. [Collect a sequence of characters](#) that are [ASCII digits](#). Let *s* be the resulting sequence.

2. If *s* is the empty string, then fail.

3. Let *length* be the number of characters in *s*.

4. Let *fraction* be the result of interpreting *s* as a base-ten integer, and then dividing that number by 10^{*length*}.

5. Increment *N* by *fraction*.

6. [Skip whitespace](#).

7. If *position* is past the end of *input*, then fail.

8. Set *next character* to the character in *input* pointed to by *position*, and advance *position* to the next character.

9. If *next character* is neither a U+0053 LATIN CAPITAL LETTER S character nor a U+0073 LATIN SMALL LETTER S character, then fail.

10. Set *units* to *seconds*.

Otherwise, run these substeps:

1. If *next character* is a [space character](#), then [skip whitespace](#), set *next character* to the character in *input* pointed to by *position*, and advance *position* to the next character.
2. If *next character* is a U+0059 LATIN CAPITAL LETTER Y character, or a U+0079 LATIN SMALL LETTER Y character, set *units* to *years* and set *M-disambiguator* to *months*.

If *next character* is a U+004D LATIN CAPITAL LETTER M character or a U+006D LATIN SMALL LETTER M character, and *M-disambiguator* is *months*, then set *units* to *months*.

If *next character* is a U+0057 LATIN CAPITAL LETTER W character or a U+0077 LATIN SMALL LETTER W character, set *units* to *weeks* and set *M-disambiguator* to *minutes*.

If *next character* is a U+0044 LATIN CAPITAL LETTER D character or a U+0064 LATIN SMALL LETTER D character, set *units* to *days* and set *M-disambiguator* to *minutes*.

If *next character* is a U+0048 LATIN CAPITAL LETTER H character or a U+0068 LATIN SMALL LETTER H character, set *units* to *hours* and set *M-disambiguator* to *minutes*.

If *next character* is a U+004D LATIN CAPITAL LETTER M character or a U+006D LATIN SMALL LETTER M character, and *M-disambiguator* is *minutes*, then set *units* to *minutes*.

If *next character* is a U+0053 LATIN CAPITAL LETTER S character or a U+0073 LATIN SMALL LETTER S character, set *units* to *seconds* and set *M-disambiguator* to *minutes*.

Otherwise if *next character* is none of the above characters, then fail.

10. Increment *component count*.
11. Let *multiplier* be 1.
12. If *units* is *years*, multiply *multiplier* by 12 and set *units* to *months*.
13. If *units* is *months*, add the product of *N* and *multiplier* to *months*.

Otherwise, run these substeps:

1. If *units* is *weeks*, multiply *multiplier* by 7 and set *units* to *days*.
2. If *units* is *days*, multiply *multiplier* by 24 and set *units* to *hours*.
3. If *units* is *hours*, multiply *multiplier* by 60 and set *units* to *minutes*.
4. If *units* is *minutes*, multiply *multiplier* by 60 and set *units* to *seconds*.
5. Forcibly, *units* is now *seconds*. Add the product of *N* and *multiplier* to *seconds*.

14. [Skip whitespace](#).

9. If *component count* is zero, fail.
10. If *months* is not zero, fail.
11. Return the [duration](#) consisting of *seconds* seconds.

§ 2.4.5.10. Vaguer moments in time

A string is a **valid date string with optional time** if it is also one of the following:

- A [valid date string](#)
- A [valid global date and time string](#)



The rules to **parse a date or time string** are as follows. The algorithm will return either a [date](#), a [time](#), a [global date and time](#), or nothing. If at any point the algorithm says that it "fails", this means that it is aborted at that point and returns nothing.

1. Let *input* be the string being parsed.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
3. Set *start position* to the same position as *position*.
4. Set the *date present* and *time present* flags to true.
5. [Parse a date component](#) to obtain *year*, *month*, and *day*. If this fails, then set the *date present* flag to false.
6. If *date present* is true, and *position* is not beyond the end of *input*, and the character at *position* is either a U+0054 LATIN CAPITAL LETTER T character (T) or a U+0020 SPACE character, then advance *position* to the next character in *input*.

Otherwise, if *date present* is true, and either *position* is beyond the end of *input* or the character at *position* is neither a U+0054 LATIN CAPITAL LETTER T character (T) nor a U+0020 SPACE character, then set *time present* to false.

Otherwise, if *date present* is false, set *position* back to the same position as *start position*.
7. If the *time present* flag is true, then [parse a time component](#) to obtain *hour*, *minute*, and *second*. If this returns nothing, then fail.
8. If the *date present* and *time present* flags are both true, but *position* is beyond the end of *input*, then fail.
9. If the *date present* and *time present* flags are both true, [parse a time-zone offset component](#) to obtain *timezone_{hours}* and *timezone_{minutes}*. If this returns nothing, then fail.
10. If *position* is not beyond the end of *input*, then fail.
11. If the *date present* flag is true and the *time present* flag is false, then let *date* be the date with year *year*, month *month*, and day *day*, and return *date*.

Otherwise, if the *time present* flag is true and the *date present* flag is false, then let *time* be the time with hour *hour*, minute *minute*, and second *second*, and return *time*.

Otherwise, let *time* be the moment in time at year *year*, month *month*, day *day*, hours *hour*, minute *minute*, second *second*, subtracting *timezone_{hours}* hours and *timezone_{minutes}* minutes, that moment in time being a moment in the UTC time zone; let *timezone* be *timezone_{hours}* hours and *timezone_{minutes}* minutes from UTC; and return *time* and *timezone*.

§ 2.4.6. Colors

A **simple color** consists of three 8-bit numbers in the range 0..255, representing the red, green, and blue components of the color respectively, in the sRGB color space. [\[SRGB\]](#)

A string is a **valid simple color** if it is exactly seven characters long, and the first character is a U+0023 NUMBER SIGN character (#), and the remaining six characters are all [ASCII hex digits](#), with the first two digits representing the red component, the middle two digits representing the green component, and the last two digits representing the blue component, in hexadecimal.

A string is a **valid lowercase simple color** if it is a [valid simple color](#) and doesn't use any characters in the range U+0041 LATIN CAPITAL LETTER A to U+0046 LATIN CAPITAL LETTER F.

The **rules for parsing simple color values** are as given in the following algorithm. When invoked, the steps must be followed in the order given, aborting at the first step that returns a value. This algorithm will return either a [simple color](#) or an error.

1. Let *input* be the string being parsed.
2. If *input* is not exactly seven characters long, then return an error.

3. If the first character in `input` is not a U+0023 NUMBER SIGN character (#), then return an error.
4. If the last six characters of `input` are not all [ASCII hex digits](#), then return an error.
5. Let `result` be a [simple color](#).
6. Interpret the second and third characters as a hexadecimal number and let the result be the red component of `result`.
7. Interpret the fourth and fifth characters as a hexadecimal number and let the result be the green component of `result`.
8. Interpret the sixth and seventh characters as a hexadecimal number and let the result be the blue component of `result`.
9. Return `result`.

The **rules for serializing simple color values** given a [simple color](#) are as given in the following algorithm:

1. Let `result` be a string consisting of a single U+0023 NUMBER SIGN character (#).
2. Convert the red, green, and blue components in turn to two-digit hexadecimal numbers using [lowercase ASCII hex digits](#), zero-padding if necessary, and append these numbers to `result`, in the order red, green, blue.
3. Return `result`, which will be a [valid lowercase simple color](#).



Some obsolete legacy attributes parse colors in a more complicated manner, using the **rules for parsing a legacy color value**, which are given in the following algorithm. When invoked, the steps must be followed in the order given, aborting at the first step that returns a value. This algorithm will return either a [simple color](#) or an error.

1. Let `input` be the string being parsed.
2. If `input` is the empty string, then return an error.
3. [Strip leading and trailing whitespace](#) from `input`.
4. If `input` is an [ASCII case-insensitive](#) match for the string "transparent", then return an error.
5. If `input` is an [ASCII case-insensitive](#) match for one of the [named colors](#), then return the [simple color](#) corresponding to that keyword. [\[CSS3COLOR\]](#)

NOTE:

[CSS2 System Colors](#) are not recognized.

6. If `input` is four characters long, and the first character in `input` is a U+0023 NUMBER SIGN character (#), and the last three characters of `input` are all [ASCII hex digits](#), then run these substeps:
 1. Let `result` be a [simple color](#).
 2. Interpret the second character of `input` as a hexadecimal digit; let the red component of `result` be the resulting number multiplied by 17.
 3. Interpret the third character of `input` as a hexadecimal digit; let the green component of `result` be the resulting number multiplied by 17.
 4. Interpret the fourth character of `input` as a hexadecimal digit; let the blue component of `result` be the resulting number multiplied by 17.
 5. Return `result`.
7. Replace any characters in `input` that have a Unicode code point greater than U+FFFF (i.e., any characters that are not in the basic multilingual plane) with the two-character string "00".
8. If `input` is longer than 128 characters, truncate `input`, leaving only the first 128 characters.
9. If the first character in `input` is a U+0023 NUMBER SIGN character (#), remove it.

10. Replace any character in *input* that is not an [ASCII hex digit](#) with the character U+0030 DIGIT ZERO (0).
11. While *input*'s length is zero or not a multiple of three, append a U+0030 DIGIT ZERO (0) character to *input*.
12. Split *input* into three strings of equal length, to obtain three components. Let *length* be the length of those components (one third the length of *input*).
13. If *length* is greater than 8, then remove the leading *length*-8 characters in each component, and let *length* be 8.
14. While *length* is greater than two and the first character in each component is a U+0030 DIGIT ZERO (0) character, remove that character and reduce *length* by one.
15. If *length* is *still* greater than two, truncate each component, leaving only the first two characters in each.
16. Let *result* be a [simple color](#).
17. Interpret the first component as a hexadecimal number; let the red component of *result* be the resulting number.
18. Interpret the second component as a hexadecimal number; let the green component of *result* be the resulting number.
19. Interpret the third component as a hexadecimal number; let the blue component of *result* be the resulting number.
20. Return *result*.

§ 2.4.7. Space-separated tokens

A **set of space-separated tokens** is a string containing zero or more words (known as tokens) separated by one or more [space characters](#), where words consist of any string of one or more characters, none of which are [space characters](#).

A string containing a [set of space-separated tokens](#) may have leading or trailing [space characters](#).

An **unordered set of unique space-separated tokens** is a [set of space-separated tokens](#) where none of the tokens are duplicated.

An **ordered set of unique space-separated tokens** is a [set of space-separated tokens](#) where none of the tokens are duplicated but where the order of the tokens is meaningful.

[Sets of space-separated tokens](#) sometimes have a defined set of allowed values. When a set of allowed values is defined, the tokens must all be from that list of allowed values; other values are non-conforming. If no such set of allowed values is provided, then all values are conforming.

NOTE:

How tokens in a [set of space-separated tokens](#) are to be compared (e.g., case-sensitively or not) is defined on a per-set basis.

When a user agent has to **split a string on spaces**, it must use the following algorithm:

1. Let *input* be the string being parsed.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
3. Let *tokens* be an ordered list of tokens, initially empty.
4. [Skip whitespace](#)
5. While *position* is not past the end of *input*:
 1. [Collect a sequence of characters](#) that are not [space characters](#).
 2. Append the string collected in the previous step to *tokens*.
 3. [Skip whitespace](#)
6. Return *tokens*.

§ 2.4.8. Comma-separated tokens

A **set of comma-separated tokens** is a string containing zero or more tokens each separated from the next by a single U+002C COMMA character (,), where tokens consist of any string of zero or more characters, neither beginning nor ending with [space characters](#), nor containing any U+002C COMMA characters (,), and optionally surrounded by [space characters](#).

EXAMPLE 51

For instance, the string " a ,b,d d " consists of four tokens: "a", "b", the empty string, and "d d". Leading and trailing whitespace around each token doesn't count as part of the token, and the empty string can be a token.

[Sets of comma-separated tokens](#) sometimes have further restrictions on what consists a valid token. When such restrictions are defined, the tokens must all fit within those restrictions; other values are non-conforming. If no such restrictions are specified, then all values are conforming.

When a user agent has to **split a string on commas**, it must use the following algorithm:

1. Let *input* be the string being parsed.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
3. Let *tokens* be an ordered list of tokens, initially empty.
4. *Token*: If *position* is past the end of *input*, jump to the last step.
5. [Collect a sequence of characters](#) that are not U+002C COMMA characters (,). Let *s* be the resulting sequence (which might be the empty string).
6. [Strip leading and trailing whitespace](#) from *s*.
7. Append *s* to *tokens*.
8. If *position* is not past the end of *input*, then the character at *position* is a U+002C COMMA character (,); advance *position* past that character.
9. Jump back to the step labeled *token*.
10. Return *tokens*.

§ 2.4.9. References

A **valid hash-name reference** to an element of type *type* is a string consisting of a U+0023 NUMBER SIGN character (#) followed by a string which exactly matches the value of the *name* attribute of an element with type *type* in the document.

The **rules for parsing a hash-name reference** to an element of type *type*, given a context node *scope*, are as follows:

1. If the string being parsed does not contain a U+0023 NUMBER SIGN character, or if the first such character in the string is the last character in the string, then return null and abort these steps.
2. Let *s* be the string from the character immediately after the first U+0023 NUMBER SIGN character in the string being parsed up to the end of that string.
3. Return the first element of type *type* in [tree order](#) in the subtree rooted at *scope* that has an *id* attribute whose value is a [case-sensitive](#) match for *s* or a *name* attribute whose value is a [compatibility caseless](#) match for *s*.

§ 2.4.10. Media queries

A string is a **valid media query list** if it matches the `<media-query-list>` production of the Media Queries specification. [\[MEDIAQ\]](#)

A string **matches the environment** of the user if it is the empty string, a string consisting of only [space characters](#), or is a media query list that matches the user's environment according to the definitions given in the Media Queries specification. [\[MEDIAQ\]](#)

§ 2.5. URLs

§ 2.5.1. Terminology

A [URL](#) is a **valid URL** if it conforms to the authoring conformance requirements in the WHATWG URL standard. [\[URL\]](#)

A string is a **valid non-empty URL** if it is a [valid URL](#) but it is not the empty string.

A string is a **valid URL potentially surrounded by spaces** if, after [stripping leading and trailing whitespace](#) from it, it is a [valid URL](#).

A string is a **valid non-empty URL potentially surrounded by spaces** if, after [stripping leading and trailing whitespace](#) from it, it is a [valid non-empty URL](#).

This specification defines the URL **about:legacy-compat** as a reserved, though unresolvable, [about:](#) URL, for use in [DOCTYPEs](#) in [HTML documents](#) when needed for compatibility with XML tools. [\[RFC6694\]](#)

This specification defines the URL **about:srcdoc** as a reserved, though unresolvable, [about:](#) URL, that is used as [the document's address](#) of [iframe srcdoc documents](#). [\[RFC6694\]](#)

The **fallback base URL** of a [Document](#) object is the [absolute URL](#) obtained by running these substeps:

1. If *document* is an [iframe srcdoc document](#), then return the [document base URL](#) of the *Document*'s [browsing context](#)'s [browsing context container](#)'s [node document](#).
2. If *document*'s [URL](#) is `about:blank`, and the *Document*'s [browsing context](#) has a [creator browsing context](#), then return the [creator base URL](#).
3. Return *document*'s [URL](#).

The **document base URL** of a [Document](#) object is the [absolute URL](#) obtained by running these substeps:

1. If there is no `<base>` element that has an [href](#) attribute in the [Document](#), then the [document base URL](#) is the [Document](#)'s [fallback base URL](#); abort these steps.
2. Otherwise, the [document base URL](#) is the [frozen base URL](#) of the first `<base>` element in the [Document](#) that has an [href](#) attribute, in [tree order](#).

§ 2.5.2. Parsing URLs

Parsing a URL is the process of taking a URL string and obtaining the [URL record](#) that it implies. While this process is defined in the WHATWG URL standard, this specification defines a wrapper for convenience. [\[URL\]](#)

NOTE:

This wrapper is only useful when the character encoding for the URL parser has to match that of the document or environment settings object for legacy reasons. When that is not the case the [URL parser](#) can be used directly.

To **parse a URL** *url*, relative to either a *document* or *environment settings object*, the user agent must use the following steps. Parsing a URL either results in failure or a [resulting URL string](#) and [resulting URL record](#).

1. Let *encoding* be *document*'s [character encoding](#), if *document* was given, and *environment settings object*'s [API URL character encoding](#) otherwise.
2. Let *baseURL* be *document*'s [base URL](#), if *document* was given, and *environment settings object*'s [API base URL](#) otherwise.
3. Let *urlRecord* be the result of applying the [URL parser](#) to *url*, with *baseURL* and *encoding*.
4. If *urlRecord* is failure, then abort these steps with an error.
5. Let *urlString* be the result of applying the [URL serializer](#) to *urlRecord*.
6. Return *urlString* as the **resulting URL string** and *urlRecord* as the **resulting URL record**.

§ 2.5.3. Dynamic changes to base URLs

When a document's [document base URL](#) changes, all elements in that document are [affected by a base URL change](#).

The following are [base URL change steps](#), which run when an element is [affected by a base URL change](#) (as defined by the DOM specification):

↪ If the element creates a [hyperlink](#)

If the [URL](#) identified by the hyperlink is being shown to the user, or if any data derived from that [URL](#) is affecting the display, then the [href](#) attribute should be [reparsed](#) relative to the element's [node document](#) and the UI updated appropriately.

EXAMPLE 52

For example, the CSS `:link/:visited` pseudo-classes might have been affected.

↪ If the element is a `<q>`, `<blockquote>`, `<ins>`, or `<del>` element with a `cite` attribute

If the [URL](#) identified by the `cite` attribute is being shown to the user, or if any data derived from that [URL](#) is affecting the display, then the [URL](#) should be [reparsed](#) relative to the element's [node document](#) and the UI updated appropriately.

↪ Otherwise

The element is not directly affected.

EXAMPLE 53

For instance, changing the base URL doesn't affect the image displayed by `img` elements, although subsequent accesses of the `src` IDL attribute from script will return a new [absolute URL](#) that might no longer correspond to the image being shown.

§ 2.6. Fetching resources

§ 2.6.1. Terminology

User agents can implement a variety of transfer protocols, but this specification mostly defines behavior in terms of HTTP. [\[HTTP\]](#)

The **HTTP GET method** is equivalent to the default retrieval action of the protocol. For example, RETR in FTP. Such actions are idempotent and safe, in HTTP terms.

The **HTTP response codes** are equivalent to statuses in other protocols that have the same basic meanings. For example, a "file not found" error is equivalent to a 404 code, a server error is equivalent to a 5xx code, and so on.

The **HTTP headers** are equivalent to fields in other protocols that have the same basic meaning. For example, the HTTP authentication headers are equivalent to the authentication aspects of the FTP protocol.

A **referrer source** is either a [Document](#) or a [URL](#).

To **create a potential-CORS request**, given a `url`, `corsAttributeState`, and an optional *same-origin fallback flag*, run these steps:

1. Let `mode` be "no-cors" if `corsAttributeState` is [No CORS](#), and "cors" otherwise.
2. If *same-origin fallback flag* is set and `mode` is "no-cors", set `mode` to "same-origin".
3. Let `credentialsMode` be "include".
4. If `corsAttributeState` is [Anonymous](#), set `credentialsMode` to "same-origin".
5. Let `request` be a new [request](#) whose [URL](#) is `url`, [destination](#) is "subresource", [mode](#) is `mode`, [credentials mode](#) is `credentialsMode`, and whose [use-URL-credentials flag](#) is set.

§ 2.6.2. Processing model

When a user agent is to [fetch](#) a resource or [URL](#), optionally **from** an origin *origin*, optionally **using** a specific [referrer source](#) as an *override referrer source*, and optionally with any of a *synchronous flag*, a *manual redirect flag*, a *force same-origin flag*, and a *block cookies flag*, the following steps must be run. (When a *URL* is to be fetched, the *URL* identifies a resource to be obtained.)

1. If there is a specific *override referrer source*, and it is a [URL](#), then let *referrer* be the *override referrer source*, and jump to the step labeled *clean referrer*.
2. Let *document* be the appropriate [Document](#) as given by the following list:
 - ↪ **If there is a specific *override referrer source***
The *override referrer source*.
 - ↪ **When [navigating](#)**
The [active document](#) of the [source browsing context](#).
 - ↪ **When fetching resources for an element**
The element's [Document](#).
3. While *document* is an [iframe srcdoc document](#), let *document* be *document*'s [browsing context](#)'s [browsing context container](#)'s [Document](#) instead.
4. If the [origin](#) of *Document* is not a scheme/host/port tuple, then set *referrer* to the empty string and jump to the step labeled *Clean referrer*.
5. Let *referrer* be [the document's address](#) of *document*.
6. *Clean referrer*: Apply the [URL parser](#) to *referrer* and let *parsed referrer* be the [resulting URL record](#).
7. Let *referrer* be the result of applying the [URL serializer](#) to *parsed referrer*, with the *exclude fragment flag* set.
8. If *referrer* is not the empty string, is not a [data: URL](#), and is not the [URL](#) "about:blank", then generate the *address of the resource from which Request-URLs are obtained* as required by HTTP for the Referer (sic) header from *referrer*. [\[HTTP\]](#)

Otherwise, the Referer (sic) header must be omitted, regardless of its value.
9. If the algorithm was not invoked with the *synchronous flag*, perform the remaining steps [in parallel](#).
10. If the [Document](#) with which any [tasks queued](#) by this algorithm would be associated doesn't have an associated [browsing context](#), then abort these steps.
11. This is the *main step*.

If the resource is to be obtained from an [application cache](#), then use the data from that [application cache](#), as if it had been obtained in the manner appropriate given its [URL](#).

If the resource is identified by an [absolute URL](#), and the resource is to be obtained using an idempotent action (such as an HTTP GET [or equivalent](#)), and it is already being downloaded for other reasons (e.g., another invocation of this algorithm), and this request would be identical to the previous one (e.g., same Accept and Origin headers), and the user agent is configured such that it is to reuse the data from the existing download instead of initiating a new one, then use the results of the existing download instead of starting a new one.

Otherwise, if the resource is identified by an [absolute URL](#) with a scheme that does not define a mechanism to obtain the resource (e.g., it is a [mailto: URL](#)) or that the user agent does not support, then act as if the resource was an HTTP 204 No Content response with no other metadata.

Otherwise, if the resource is identified by the [URL](#) [about:blank](#), then the resource is immediately available and consists of the empty string, with no metadata.

Otherwise, at a time convenient to the user and the user agent, download (or otherwise obtain) the resource, applying the semantics of the relevant specifications (e.g., performing an HTTP GET or POST operation, or reading the file from disk, or expanding [data: URLs](#), etc).

For the purposes of the Referer (sic) header, use the *address of the resource from which Request-URLs are obtained* generated in the earlier step.

For the purposes of the Origin header, if the [fetching algorithm](#) was explicitly initiated from an *origin*, then *the origin that initiated the HTTP request is origin*. Otherwise, this is a request from a "privacy-sensitive" context. [\[ORIGIN\]](#)

12. If the algorithm was not invoked with the *block cookies flag*, and there are cookies to be set, update the cookies.

[\[COOKIES\]](#) 

13. If the fetched resource is an HTTP redirect [or equivalent](#), then:

↪ **If the *force same-origin flag* is set and the [URL](#) of the target of the redirect does not have the [same origin](#) as the [URL](#) for which the [fetch](#) algorithm was invoked**

Abort these steps and return failure from this algorithm, as if the remote host could not be contacted.

↪ **If the *manual redirect flag* is set**

Continue, using the fetched resource (the redirect) as the result of the algorithm. If the calling algorithm subsequently requires the user agent to **transparently follow the redirect**, then the user agent must resume this algorithm from the *main step*, but using the target of the redirect as the resource to fetch, rather than the original resource.

↪ **Otherwise**

First, apply any relevant requirements for redirects (such as showing any appropriate prompts). Then, redo *main step*, but using the target of the redirect as the resource to fetch, rather than the original resource. For HTTP requests, the new request must include the same headers as the original request, except for headers for which other requirements are specified (such as the Host header). [\[HTTP\]](#)

NOTE:

The HTTP specification requires that 301, 302, and 307 redirects, when applied to methods other than the safe methods, not be followed without user confirmation. That would be an appropriate prompt for the purposes of the requirement in the paragraph above. [\[HTTP\]](#)

14. If the algorithm was not invoked with the *synchronous flag*: When the resource is available, or if there is an error of some description, [queue a task](#) that uses the resource as appropriate. If the resource can be processed incrementally, as, for instance, with a progressively interlaced JPEG or an HTML file, additional tasks may be queued to process the data as it is downloaded. The [task source](#) for these [tasks](#) is the [networking task source](#).

Otherwise, return the resource or error information to the calling algorithm.

If the user agent can determine the actual length of the resource being [fetched](#) for an instance of this algorithm, and if that length is finite, then that length is the file's **size**. Otherwise, the subject of the algorithm (that is, the resource being fetched) has no known [size](#). (For example, the HTTP Content-Length header might provide this information.)

The user agent must also keep track of the **number of bytes downloaded** for each instance of this algorithm. This number must exclude any out-of-band metadata, such as HTTP headers.

NOTE:

The [navigation](#) processing model handles redirects itself, overriding the redirection handling that would be done by the [fetching algorithm](#).

NOTE:

Whether the [type sniffing rules](#) apply to the fetched resource depends on the algorithm that invokes the rules — they are not always applicable.

§ 2.6.3. Encrypted HTTP and related security concerns

Anything in this specification that refers to HTTP also applies to HTTP-over-TLS, as represented by [URLs](#) representing the https scheme. [\[HTTP\]](#)

⚠Warning! User agents should report certificate errors to the user and must either refuse to download resources sent with erroneous certificates or must act as if such resources were in fact served with no encryption.

User agents should warn the user that there is a potential problem whenever the user visits a page that the user has previously visited, if the page uses less secure encryption on the second visit.

Not doing so can result in users not noticing man-in-the-middle attacks.

EXAMPLE 54

If a user connects to a server with a self-signed certificate, the user agent could allow the connection but just act as if there had been no encryption. If the user agent instead allowed the user to override the problem and then displayed the page as if it was fully and safely encrypted, the user could be easily tricked into accepting man-in-the-middle connections.

If a user connects to a server with full encryption, but the page then refers to an external resource that has an expired certificate, then the user agent will act as if the resource was unavailable, possibly also reporting the problem to the user. If the user agent instead allowed the resource to be used, then an attacker could just look for "secure" sites that used resources from a different host and only apply man-in-the-middle attacks to that host, for example taking over scripts in the page.

If a user bookmarks a site that uses a CA-signed certificate, and then later revisits that site directly but the site has started using a self-signed certificate, the user agent could warn the user that a man-in-the-middle attack is likely underway, instead of simply acting as if the page was not encrypted.

§ 2.6.4. Determining the type of a resource

The **Content-Type metadata** of a resource must be obtained and interpreted in a manner consistent with the requirements of the MIME Sniffing specification. [\[MIMESNIFF\]](#)

The **computed type of a resource** must be found in a manner consistent with the requirements given in the MIME Sniffing specification for finding the *computed media type* of the relevant sequence of octets. [\[MIMESNIFF\]](#)

The **rules for sniffing images specifically** and the **rules for distinguishing if a resource is text or binary** are also defined in the MIME Sniffing specification. Both sets of rules return a [MIME type](#) as their result. [\[MIMESNIFF\]](#)

⚠Warning! It is imperative that the rules in the MIME Sniffing specification be followed exactly. When a user agent uses different heuristics for content type detection than the server expects, security problems can occur. For more details, see the MIME Sniffing specification. [\[MIMESNIFF\]](#)

§ 2.6.5. Extracting character encodings from `<meta>` elements

The **algorithm for extracting a character encoding from a `<meta>` element**, given a string `s`, is as follows. It either returns a character encoding or nothing.

1. Let *position* be a pointer into `s`, initially pointing at the start of the string.
2. *Loop*: Find the first seven characters in `s` after *position* that are an [ASCII case-insensitive](#) match for the word "charset". If no such match is found, return nothing and abort these steps.
3. Skip any [space characters](#) that immediately follow the word "charset" (there might not be any).
4. If the next character is not a U+003D EQUALS SIGN (=), then move *position* to point just before that next character, and jump back to the step labeled *loop*.
5. Skip any [space characters](#) that immediately follow the equals sign (there might not be any).
6. Process the next character as follows:

- ↪ If it is a U+0022 QUOTATION MARK character (") and there is a later U+0022 QUOTATION MARK character (") in *s*
- ↪ If it is a U+0027 APOSTROPHE character (') and there is a later U+0027 APOSTROPHE character (') in *s*

Return the result of [getting an encoding](#) from the substring that is between this character and the next earliest occurrence of this character.

- ↪ If it is an unmatched U+0022 QUOTATION MARK character (")
- ↪ If it is an unmatched U+0027 APOSTROPHE character (')
- ↪ If there is no next character

Return nothing.

- ↪ Otherwise

Return the result of [getting an encoding](#) from the substring that consists of this character up to but not including the first [space character](#) or U+003B SEMICOLON character (;), or the end of *s*, whichever comes first.

NOTE:

This algorithm is distinct from those in the HTTP specification (for example, HTTP doesn't allow the use of single quotes and requires supporting a backslash-escape mechanism that is not supported by this algorithm). While the algorithm is used in contexts that, historically, were related to HTTP, the syntax as supported by implementations diverged some time ago. [\[HTTP\]](#)

§ 2.6.6. CORS settings attributes

A **CORS settings attribute** is an [enumerated attribute](#). The following table lists the keywords and states for the attribute — the keywords in the left column map to the states in the cell in the second column on the same row as the keyword.

Keyword	State	Brief description
'anonymous'	Anonymous	Requests for the element will have their mode set to "cors" and their credentials mode set to "same-origin".
'use-credentials'	Use Credentials	Requests for the element will have their mode set to "cors" and their credentials mode set to "include".

The empty string is also a valid keyword, and maps to the [Anonymous](#) state. The attribute's *invalid value default* is the [Anonymous](#) state. For the purposes of [reflection](#), the canonical case for the [Anonymous](#) state is the 'anonymous' keyword. The *missing value default*, used when the attribute is omitted, is the **No CORS** state.

§ 2.7. Common DOM interfaces

§ 2.7.1. Reflecting content attributes in IDL attributes

Some IDL attributes are defined to **reflect** a particular content attribute. This means that on getting, the IDL attribute returns the current value of the content attribute, and on setting, the IDL attribute changes the value of the content attribute to the given value.

In general, on getting, if the content attribute is not present, the IDL attribute must act as if the content attribute's value is the empty string; and on setting, if the content attribute is not present, it must first be added.

If a reflecting IDL attribute is a DOMString attribute whose content attribute is defined to contain a [URL](#), then on getting, the IDL attribute must [parse](#) the value of the content attribute relative to the element and return the resulting [absolute URL](#) if that was successful, or the empty string otherwise; and on setting, must set the content attribute to the specified literal value. If the content attribute is absent, the IDL attribute must return the default value, if the content attribute has one, or else the empty string.

If a reflecting IDL attribute is a DOMString attribute whose content attribute is defined to contain one or more [URLs](#), then on getting, the IDL attribute must [split the content attribute on spaces](#) and return the concatenation of [parsing](#) each token URL to an [absolute URL](#) relative to the element, with a single U+0020 SPACE character between each URL, ignoring any tokens that did not resolve successfully. If the content attribute is absent, the IDL attribute must return the default value, if

the content attribute has one, or else the empty string. On setting, the IDL attribute must set the content attribute to the specified literal value.

If a reflecting IDL attribute is a DOMString attribute whose content attribute is an [enumerated attribute](#), and the IDL attribute is **limited to only known values**, then, on getting, the IDL attribute must return the conforming value associated with the state the attribute is in (in its canonical case), if any, or the empty string if the attribute is in a state that has no associated keyword value or if the attribute is not in a defined state (e.g., the attribute is missing and there is no *missing value default*); and on setting, the content attribute must be set to the specified new value.

If a reflecting IDL attribute is a nullable DOMString attribute whose content attribute is an [enumerated attribute](#), then, on getting, if the corresponding content attribute is in its *missing value default* then the IDL attribute must return null, otherwise, the IDL attribute must return the conforming value associated with the state the attribute is in (in its canonical case); and on setting, if the new value is null, the content attribute must be removed, and otherwise, the content attribute must be set to the specified new value.

If a reflecting IDL attribute is a DOMString attribute but doesn't fall into any of the above categories, then the getting and setting must be done in a transparent, case-preserving manner.

If a reflecting IDL attribute is a boolean attribute, then on getting the IDL attribute must return true if the content attribute is set, and false if it is absent. On setting, the content attribute must be removed if the IDL attribute is set to false, and must be set to the empty string if the IDL attribute is set to true. (This corresponds to the rules for [boolean content attributes](#).)

If a reflecting IDL attribute has a signed integer type (`long`) then, on getting, the content attribute must be parsed according to the [rules for parsing signed integers](#), and if that is successful, and the value is in the range of the IDL attribute's type, the resulting value must be returned. If, on the other hand, it fails or returns an out of range value, or if the attribute is absent, then the default value must be returned instead, or 0 if there is no default value. On setting, the given value must be converted to the shortest possible string representing the number as a [valid integer](#) and then that string must be used as the new content attribute value.

If a reflecting IDL attribute has a signed integer type (`long`) that is **limited to only non-negative numbers** then, on getting, the content attribute must be parsed according to the [rules for parsing non-negative integers](#), and if that is successful, and the value is in the range of the IDL attribute's type, the resulting value must be returned. If, on the other hand, it fails or returns an out of range value, or if the attribute is absent, the default value must be returned instead, or -1 if there is no default value. On setting, if the value is negative, the user agent must throw an `IndexSizeError` exception. Otherwise, the given value must be converted to the shortest possible string representing the number as a [valid non-negative integer](#) and then that string must be used as the new content attribute value.

If a reflecting IDL attribute has an *unsigned* integer type (`unsigned long`) then, on getting, the content attribute must be parsed according to the [rules for parsing non-negative integers](#), and if that is successful, and the value is in the range 0 to 2147483647 inclusive, the resulting value must be returned. If, on the other hand, it fails or returns an out of range value, or if the attribute is absent, the default value must be returned instead, or 0 if there is no default value. On setting, first, if the new value is in the range 0 to 2147483647, then let n be the new value, otherwise let n be the default value, or 0 if there is no default value; then, n must be converted to the shortest possible string representing the number as a [valid non-negative integer](#) and that string must be used as the new content attribute value.

If a reflecting IDL attribute has an unsigned integer type (`unsigned long`) that is **limited to only non-negative numbers greater than zero**, then the behavior is similar to the previous case, but zero is not allowed. On getting, the content attribute must first be parsed according to the [rules for parsing non-negative integers](#), and if that is successful, and the value is in the range 1 to 2147483647 inclusive, the resulting value must be returned. If, on the other hand, it fails or returns an out of range value, or if the attribute is absent, the default value must be returned instead, or 1 if there is no default value. On setting, if the value is zero, the user agent must throw an `IndexSizeError` exception. Otherwise, first, if the new value is in the range 1 to 2147483647, then let n be the new value, otherwise let n be the default value, or 1 if there is no default value; then, n must be converted to the shortest possible string representing the number as a [valid non-negative integer](#) and that string must be used as the new content attribute value.

If a reflecting IDL attribute has a floating-point number type (`double` or `unrestricted double`), then, on getting, the content attribute must be parsed according to the [rules for parsing floating-point number values](#), and if that is successful, the resulting value must be returned. If, on the other hand, it fails, or if the attribute is absent, the default value must be returned instead, or 0.0 if there is no default value. On setting, the given value must be converted to the [best representation of the number as a floating-point number](#) and then that string must be used as the new content attribute value.

If a reflecting IDL attribute has a floating-point number type (double or unrestricted double) that is **limited to numbers greater than zero**, then the behavior is similar to the previous case, but zero and negative values are not allowed. On getting, the content attribute must be parsed according to the [rules for parsing floating-point number values](#), and if that is successful and the value is greater than 0.0, the resulting value must be returned. If, on the other hand, it fails or returns an out of range value, or if the attribute is absent, the default value must be returned instead, or 0.0 if there is no default value. On setting, if the value is less than or equal to zero, then the value must be ignored. Otherwise, the given value must be converted to the [best representation of the number as a floating-point number](#) and then that string must be used as the new content attribute value.

NOTE:

The values Infinity and Not-a-Number (NaN) values throw an exception on setting, as defined in the Web IDL specification. [\[WEBIDL\]](#)

If a reflecting IDL attribute has the type [DOMTokenList](#), then on getting it must return a [DOMTokenList](#) object whose associated element is the element in question and whose associated attribute's local name is the name of the attribute in question. The same [DOMTokenList](#) object must be returned every time for each attribute.

If a reflecting IDL attribute has the type [HTMLElement](#), or an interface that descends from [HTMLElement](#), then, on getting, it must run the following algorithm (stopping at the first point where a value is returned):

1. If the corresponding content attribute is absent, then the IDL attribute must return null.
2. Let *candidate* be the element that the `document.getElementById()` method would find when called on the content attribute's element's [node document](#) if it were passed as its argument the current value of the corresponding content attribute.
3. If *candidate* is null, or if it is not type-compatible with the IDL attribute, then the IDL attribute must return null.
4. Otherwise, it must return *candidate*.

On setting, if the given element has an `id` attribute, and has the same [home subtree](#) as the element of the attribute being set, and the given element is the first element in that [home subtree](#) whose [ID](#) is the value of that `id` attribute, then the content attribute must be set to the value of that `id` attribute. Otherwise, the content attribute must be set to the empty string.

§ 2.7.2. Collections

The [HTMLFormControlsCollection](#) and [HTMLOptionsCollection](#) interfaces are [collections](#) derived from the [HTMLCollection](#) interface. The [HTMLAllCollection](#) however, is independent as it has a variety of unique quirks that are not desirable to inherit from [HTMLCollection](#).

§ 2.7.2.1. The HTMLAllCollection interface

The [HTMLAllCollection](#) interface is used for the legacy `document.all` attribute. It operates similarly to [HTMLCollection](#); it also supports a variety of other legacy features required for web compatibility such as the ability to be invoked like a function ([legacycaller](#)).

NOTE:

All [HTMLAllCollection](#) objects are rooted at a [Document](#) and have a filter that matches all elements, so the elements [represented by the collection](#) of an [HTMLAllCollection](#) object consist of all the descendant elements of the root [Document](#).

```
[LegacyUnenumerableNamedProperties]
interface HTMLAllCollection {
  readonly attribute unsigned long length;
  getter Element? (unsigned long index);
  getter (HTMLCollection or Element)? namedItem(DOMString name);
  legacycaller (HTMLCollection or Element)? item(optional DOMString nameOrItem);
};
```

This definition is non-normative. Implementation requirements are given below this definition.

`collection . length`

Returns the number of elements in the collection.

`element` = `collection . item(index)`

`element` = `collection(index)`

`element` = `collection[index]`

Returns the item with index `index` from the collection (determined by [tree order](#)).

`element` = `collection . item(name)`

`collection` = `collection . item(name)`

`element` = `collection . namedItem(name)`

`collection` = `collection . namedItem(name)`

`element` = `collection(name)`

`collection` = `collection(name)`

`element` = `collection[name]`

`collection` = `collection[name]`

Returns the item with [ID](#) or name `name` from the collection.

If there are multiple matching items, then an [HTMLCollection](#) object containing all those elements is returned.

The `name` attribute's value provides a name for `<button>`, `<input>`, `<select>`, and `<textarea>`. Similarly, `<iframe>`'s `name`, `<object>`'s `name`, `<meta>`'s `name`, `<map>`'s `name`, and `<form>`'s `name` attribute's value provides a name for their respective elements. Only the elements mentioned have a **name** for the purpose of this method.

The object's [supported property indices](#) are as defined for [HTMLCollection](#) objects.

The [supported property names](#) consist of the non-empty values of all the `id` and `name` attributes of all the elements [represented by the collection](#), in [tree order](#), ignoring later duplicates, with the `id` of an element preceding its `name` if it contributes both, they differ from each other, and neither is the duplicate of an earlier entry.

On getting, the `length` attribute must return the number of nodes [represented by the collection](#).

The indexed property getter must return the result of [getting the "all"-indexed element](#) from this [HTMLAllCollection](#) given the passed index.

The `namedItem(name)` method must return the result of [getting the "all"-named element or elements](#) from this [HTMLAllCollection](#) given `name`.

The `item(nameOrIndex)` method (and the legacy caller behavior) must act according to the following algorithm:

1. If `nameOrIndex` was not provided, return null.
2. If `nameOrIndex`, [converted](#) to a JavaScript string value, is an [array index property name](#), return the result of [getting the "all"-indexed element](#) from this [HTMLAllCollection](#) given the number represented by `nameOrIndex`.
3. Return the result of [getting the "all"-named element or elements](#) from this [HTMLAllCollection](#) given `nameOrIndex`.

The following elements are considered **"all"-named elements**: `<a>`, `<applet>`, `<button>`, `<embed>`, `<form>`, `<frame>`, `<frameset>`, `<iframe>`, `<img>`, `<input>`, `<map>`, `<meta>`, `<object>`, `<select>`, and `<textarea>`.

To [get the "all"-indexed element](#) from an [HTMLAllCollection](#) `collection` given an index `index`, return the element with index `index` in `collection`, or null if there is no such element at `index`.

To [get the "all"-named element or elements](#) from an [HTMLAllCollection](#) `collection` given a name `name`, run the following algorithm:

1. If `name` is the empty string, return null.
2. Let `subCollection` be an [HTMLCollection](#) object rooted at the same [Document](#) as `collection`, whose filter matches only elements that are either:

- ["all"-named elements](#) with a [name](#) attribute equal to `name`, or,
 - elements with an [ID](#) equal to `name`.
3. If there is exactly one element in `subCollection`, then return that element.
 4. Otherwise, if `subCollection` is empty, return null.
 5. Otherwise, return `subCollection`.

§ 2.7.2.2. The `HTMLFormControlsCollection` interface

The `HTMLFormControlsCollection` interface is used for [collections](#) of [listed elements](#) in `<form>` elements.

```
interface HTMLFormControlsCollection : HTMLCollection {
  // inherits length and item()
  getter (RadioNodeList or Element)? namedItem(DOMString name); // shadows inherited namedItem()
};

interface RadioNodeList : NodeList {
  attribute DOMString value;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

`collection . length`

Returns the number of elements in the collection.

`element = collection . item(index)`

`element = collection [ index ]`

Returns the item with index `index` from the collection. The items are sorted in [tree order](#).

`element = collection . namedItem(name)`

`radioNodeList = collection . namedItem(name)`

`element = collection [ name ]`

`radioNodeList = collection [ name ]`

Returns the item with [ID](#) or name `name` from the collection.

If there are multiple matching items, then a `RadioNodeList` object containing all those elements is returned.

`radioNodeList . value [ = value ]`

Returns the value of the first checked radio button represented by the object.

Can be set, to check the first radio button with the given value represented by the object.

The object's [supported property indices](#) are as defined for `HTMLCollection` objects.

The [supported property names](#) consist of the non-empty values of all the `id` and `name` attributes of all the elements [represented by the collection](#), in [tree order](#), ignoring later duplicates, with the `id` of an element preceding its name if it contributes both, they differ from each other, and neither is the duplicate of an earlier entry.

The properties exposed in this way must be [unenumerable](#).

The `namedItem(name)` method must act according to the following algorithm:

1. If `name` is the empty string, return null and stop the algorithm.
2. If, at the time the method is called, there is exactly one node in the collection that has either an `id` attribute or a `name` attribute equal to `name`, then return that node and stop the algorithm.
3. Otherwise, if there are no nodes in the collection that have either an `id` attribute or a `name` attribute equal to `name`, then return null and stop the algorithm.

4. Otherwise, create a new `RadioNodeList` object representing a [live](#) view of the `HTMLFormControlsCollection` object, further filtered so that the only nodes in the `RadioNodeList` object are those that have either an `id` attribute or a `name` attribute equal to `name`. The nodes in the `RadioNodeList` object must be sorted in [tree order](#).
5. Return that `RadioNodeList` object.



Members of the `RadioNodeList` interface inherited from the `NodeList` interface must behave as they would on a `NodeList` object.

The [value](#) IDL attribute on the `RadioNodeList` object, on getting, must return the value returned by running the following steps:

1. Let `element` be the first element in [tree order](#) represented by the `RadioNodeList` object that is an `<input>` element whose `type` attribute is in the [Radio Button](#) state and whose [checkedness](#) is true. Otherwise, let it be null.
2. If `element` is null, return the empty string.
3. If `element` is an element with no `value` attribute, return the string "on".
4. Otherwise, return the value of `element`'s `value` attribute.

On setting, the `value` IDL attribute must run the following steps:

1. If the new value is the string "on": let `element` be the first element in [tree order](#) represented by the `RadioNodeList` object that is an `<input>` element whose `type` attribute is in the [Radio Button](#) state and whose `value` content attribute is either absent, or present and equal to the new value, if any. If no such element exists, then instead let `element` be null.

Otherwise: let `element` be the first element in [tree order](#) represented by the `RadioNodeList` object that is an `<input>` element whose `type` attribute is in the [Radio Button](#) state and whose `value` content attribute is present and equal to the new value, if any. If no such element exists, then instead let `element` be null.

2. If `element` is not null, then set its [checkedness](#) to true.

2.7.2.3. The `HTMLOptionsCollection` interface

The `HTMLOptionsCollection` interface is used for [collections](#) of `<option>` elements. It is always rooted on a `<select>` element and has attributes and methods that manipulate that element's descendants.

```
interface HTMLOptionsCollection : HTMLCollection {
  // inherits item(), namedItem()
  attribute unsigned long length; // shadows inherited length
  setter void (unsigned long index, HTMLOptionElement? option);
  void add((HTMLOptionElement or HTMLOptGroupElement) element, optional (HTMLElement or long)? before = null);
  void remove(long index);
  attribute long selectedIndex;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

`collection.length` [= `value`]

Returns the number of elements in the collection.

When set to a smaller number, truncates the number of `<option>` elements in the corresponding container.

When set to a greater number, adds new blank `<option>` elements to that container.

`element` = `collection.item(index)`

`element` = `collection[index]`

Returns the item with index `index` from the collection. The items are sorted in [tree order](#).

`collection[index] = element`

When `index` is a greater number than the number of items in the collection, adds new blank `<option>` elements in the corresponding container.

When set to null, removes the item at index `index` from the collection.

When set to an `<option>` element, adds or replaces it at index `index` from the collection.

`element = collection.namedItem(name)`

`element = collection[name]`

Returns the item with `ID` or name `name` from the collection.

If there are multiple matching items, then the first is returned.

`collection.add(element [, before ])`

Inserts `element` before the node given by `before`.

The `before` argument can be a number, in which case `element` is inserted before the item with that number, or an element from the collection, in which case `element` is inserted before that element.

If `before` is omitted, null, or a number out of range, then `element` will be added at the end of the list.

This method will throw a `HierarchyRequestError` exception if `element` is an ancestor of the element into which it is to be inserted.

`collection.remove(index)`

Removes the item with index `index` from the collection.

`collection.selectedIndex [ = value ]`

Returns the index of the first selected item, if any, or -1 if there is no selected item.

Can be set, to change the selection.

The object's [supported property indices](#) are as defined for `HTMLCollection` objects.

On getting, the **length** attribute must return the number of nodes [represented by the collection](#).

On setting, the behavior depends on whether the new value is equal to, greater than, or less than the number of nodes [represented by the collection](#) at that time. If the number is the same, then setting the attribute must do nothing. If the new value is greater, then `n` new `<option>` elements with no attributes and no child nodes must be appended to the `<select>` element on which the `HTMLOptionsCollection` is rooted, where `n` is the difference between the two numbers (new value minus old value). Mutation events must be fired as if a `DocumentFragment` containing the new option elements had been inserted. If the new value is lower, then the last `n` nodes in the collection must be removed from their parent nodes, where `n` is the difference between the two numbers (old value minus new value).

NOTE:

Setting **length** never removes or adds any `<optgroup>` elements, and never adds new children to existing `<optgroup>` elements (though it can remove children from them).

The [supported property names](#) consist of the non-empty values of all the `id` and `name` attributes of all the elements [represented by the collection](#), in [tree order](#), ignoring later duplicates, with the `id` of an element preceding its name if it contributes both, they differ from each other, and neither is the duplicate of an earlier entry.

The properties exposed in this way must be [unenumerable](#).

When the user agent is to **set the value of a new indexed property** or **set the value of an existing indexed property** for a given property index `index` to a new value `value`, it must run the following algorithm:

1. If `value` is null, invoke the steps for the `remove` method with `index` as the argument, and abort these steps.
2. Let `length` be the number of nodes [represented by the collection](#).
3. Let `n` be `index` minus `length`.
4. If `n` is greater than zero, then [append](#) a `DocumentFragment` consisting of `n-1` new `<option>` elements with no attributes

and no child nodes to the [<select>](#) element on which the `HTMLOptionsCollection` is rooted.

5. If `n` is greater than or equal to zero, [append](#) `value` to the [<select>](#) element. Otherwise, [replace](#) the `index`th element in the collection by `value`.

The `add( element , before )` method must act according to the following algorithm:

1. If `element` is an ancestor of the [<select>](#) element on which the `HTMLOptionsCollection` is rooted, then throw a `HierarchyRequestError` exception and abort these steps.
2. If `before` is an element, but that element isn't a descendant of the [<select>](#) element on which the `HTMLOptionsCollection` is rooted, then throw a `NotFoundError` exception and abort these steps.
3. If `element` and `before` are the same element, then return and abort these steps.
4. If `before` is a node, then let `reference` be that node. Otherwise, if `before` is an integer, and there is a `before`th node in the collection, let `reference` be that node. Otherwise, let `reference` be null.
5. If `reference` is not null, let `parent` be the parent node of `reference`. Otherwise, let `parent` be the [<select>](#) element on which the `HTMLOptionsCollection` is rooted.
6. Act as if the DOM `insertBefore()` method was invoked on the `parent` node, with `element` as the first argument and `reference` as the second argument.

The `remove( index )` method must act according to the following algorithm:

1. If the number of nodes [represented by the collection](#) is zero, abort these steps.
2. If `index` is not a number greater than or equal to 0 and less than the number of nodes [represented by the collection](#), abort these steps.
3. Let `element` be the `index`th element in the collection.
4. Remove `element` from its parent node.

The `selectedIndex` IDL attribute must act like the identically named attribute on the [<select>](#) element on which the `HTMLOptionsCollection` is rooted

§ 2.7.3. The `DOMStringMap` interface

The `DOMStringMap` interface represents a set of name-value pairs. It exposes these using the scripting language's native mechanisms for property access.

When a `DOMStringMap` object is instantiated, it is associated with three algorithms, one for getting the list of name-value pairs, one for setting names to certain values, and one for deleting names.

```
[OverrideBuiltins]
interface DOMStringMap {
  getter DOMString (DOMString name);
  setter void (DOMString name, DOMString value);
  deleter void (DOMString name);
};
```

The [supported property names](#) on a `DOMStringMap` object at any instant are the names of each pair returned from the algorithm for getting the list of name-value pairs at that instant, in the order returned.

To [determine the value of a named property](#) `name` in a `DOMStringMap`, the user agent must return the value component of the name-value pair whose name component is `name` in the list returned by the algorithm for getting the list of name-value pairs.

To [set](#) the value of a named property `name` to value `value`, the algorithm for setting names to certain values must be run, passing `name` as the name and `value` as the value.

To **delete an existing named property** *name*, the algorithm for deleting names must be run, passing *name* as the name.

NOTE:

The DOMStringMap interface definition here is only intended for JavaScript environments. Other language bindings will need to define how DOMStringMap is to be implemented for those languages.

EXAMPLE 55

The dataset attribute on elements exposes the data-* attributes on the element.

Given the following fragment and elements with similar constructions:

```

```

...one could imagine a function splashDamage() that takes some arguments, the first of which is the element to process:

```
function splashDamage(node, x, y, damage) {
  if (node.classList.contains('tower') && // checking the 'class' attribute
      node.dataset.x == x && // reading the 'data-x' attribute
      node.dataset.y == y) { // reading the 'data-y' attribute
    var hp = parseInt(node.dataset.hp); // reading the 'data-hp' attribute
    hp = hp - damage;
    if (hp < 0) {
      hp = 0;
      node.dataset.ai = 'dead'; // setting the 'data-ai' attribute
      delete node.dataset.ability; // removing the 'data-ability' attribute
    }
    node.dataset.hp = hp; // setting the 'data-hp' attribute
  }
}
```

§ 2.7.4. The DOMELEMENTMap interface

The DOMELEMENTMap interface represents a set of name-element mappings. It exposes these using the scripting language's native mechanisms for property access.

When a DOMELEMENTMap object is instantiated, it is associated with three algorithms, one for getting the list of name-element mappings, one for mapping a name to a certain element, and one for deleting mappings by name.

```
interface DOMELEMENTMap {
  getter Element (DOMString name);
  setter creator void (DOMString name, Element value);
  deleter void (DOMString name);
};
```

The [supported property names](#) on a DOMELEMENTMap object at any instant are the names for each mapping returned from the algorithm for getting the list of name-element mappings at that instant, in the order returned.

To [determine the value of a named property](#) *name* in a DOMELEMENTMap, the user agent must return the element component of the name-element mapping whose name component is *name* in the list returned by the algorithm for getting the list of name-element mappings.

To set the value of a **new** or **existing** named property *name* to value *value*, the algorithm for mapping a name to a certain element must be run, passing *name* as the name *value* as the element.

To [delete an existing named property](#) *name*, the algorithm for deleting mappings must be run, passing *name* as the name component of the mapping to be deleted.

NOTE:

The `DOMElementMap` interface definition here is only intended for JavaScript environments. Other language bindings will need to define how `DOMElementMap` is to be implemented for those languages.

§ 2.7.5. Garbage collection

There is an **implied strong reference** from any IDL attribute that returns a pre-existing object to that object.

EXAMPLE 56

For example, the `window.document` attribute on the [Window](#) object means that there is a strong reference from a [Window](#) object to its [Document](#) object. Similarly, there is always a strong reference from a [Document](#) to any descendant nodes, and from any node to its owner [node document](#).

§ 2.8. Namespaces

The **HTML namespace** is: `http://www.w3.org/1999/xhtml`

The **MathML namespace** is: `http://www.w3.org/1998/Math/MathML`

The **SVG namespace** is: `http://www.w3.org/2000/svg`

The **XLink namespace** is: `http://www.w3.org/1999/xlink`

The **XML namespace** is: `http://www.w3.org/XML/1998/namespace`

The **XMLNS namespace** is: `http://www.w3.org/2000/xmlns/`



Data mining tools and other user agents that perform operations on content without running scripts, evaluating CSS or XPath expressions, or otherwise exposing the resulting DOM to arbitrary content, may "support namespaces" by just asserting that their DOM node analogs are in certain namespaces, without actually exposing the above strings.

**NOTE:**

In [the HTML syntax](#), namespace prefixes and namespace declarations do not have the same effect as in XML. For instance, the colon has no special meaning in HTML element names.

§ 2.9. Safe passing of structured data

This section uses the terminology and typographic conventions from the JavaScript specification. [\[ECMA-262\]](#)

§ 2.9.1. Cloneable objects

[Cloneable objects](#) support being cloned across [event loops](#). That is, they support being cloned across [Document](#) and [Worker](#) boundaries, including across [Documents](#) of different [origins](#). Not all objects are [cloneable objects](#) and not all aspects of objects that are [cloneable objects](#) are necessarily preserved when cloned.

[Platform objects](#) have the following internal method:

`[[Clone]] ( targetRealm , memory )`

Unless specified otherwise, invoking the `[[Clone]]` internal method must throw a `"DataCloneError"` [DOMException](#). (By default, [platform objects](#) are not [cloneable objects](#).)

[Platform objects](#) that are [cloneable objects](#) have a `[[Clone]]` internal method which is specified to run a series of steps. The result of running those steps must be a thrown exception or a clone of *this*, created in *targetRealm*. It is up such objects to define what cloning means for them.

Objects defined in the JavaScript specification are handled by the [StructuredClone](#) abstract operation directly.

§ 2.9.2. Transferable objects

[Transferable objects](#) support being transferred across [event loops](#). Transferring is effectively recreating the object while sharing a reference to the underlying data and then detaching the object being transferred. This is useful to transfer ownership of expensive resources. Not all objects are [transferable objects](#) and not all aspects of objects that are [transferable objects](#) are necessarily preserved when transferred.

NOTE:

Transferring is an irreversible and non-idempotent operation. Once an object has been transferred, it cannot be transferred, or indeed used, again.

[Platform objects](#) that are [transferable objects](#) have a `[[Detached]]` internal slot and the following internal method:

```
[[Transfer]] ( targetRealm )
```

NOTE:

Whereas all [platform objects](#) have a `[[Clone]]` internal method, not all have a `[[Detached]]` internal slot and a `[[Transfer]]` internal method.

[Platform objects](#) that are [transferable objects](#) must define the `[[Transfer]]` internal method such that it either throws an exception or returns a clone of *this*, created in *targetRealm*, with *this*'s underlying data shared with the return value, and *this*'s `[[Detached]]` internal slot value set to true. It is up to such objects to define what transferring means for them.

Objects defined in the JavaScript specification are handled by the [StructuredCloneWithTransfer](#) abstract operation directly. (Technically, by [IsTransferable](#) and [TransferHelper](#).)

§ 2.9.3. StructuredCloneWithTransfer (*input*, *transferList*, *targetRealm*)

1. Let *memory* be an empty map.

NOTE:

The purpose of the memory map, both here and in the [StructuredClone](#) abstract operation, is to avoid cloning objects twice. This ends up preserving cycles and the identity of duplicate objects in graphs.

2. For each object *transferable* in *transferList*:

1. If [IsTransferable](#)(*transferable*) is false, then throw a "DataCloneError" [DOMException](#).
2. Let *placeholder* be a user-agent-defined placeholder object.
3. Create an entry in *memory* with key *transferable* and value *placeholder*.

3. Let *clone* be the result of ? [StructuredClone](#)(*input*, *targetRealm*, *memory*).

4. Let *outputTransferList* be a new empty [List](#).

5. For each object *transferable* in *transferList*:

1. Let *placeholderResult* be the value of the entry in *memory* whose key is *transferable*.
2. Let *transferResult* be ? [TransferHelper](#)(*transferable*, *targetRealm*).
3. Within *clone*, replace references to *placeholderResult* with *transferResult*, such that everything holding a reference to *placeholderResult*, now holds a reference to *transferResult*.

NOTE:

This is a rather unusual low-level operation for which no primitives are defined by JavaScript.

4. Add *transferResult* as the last element of *outputTransferList*.
6. Return { *[[Clone]]*: *clone*, *[[transferList]]*: *outputTransferList* }.

NOTE:

Originally the [StructuredCloneWithTransfer](#) abstract operation was known as the "structured clone" algorithm. The [StructuredClone](#) abstract operation was known as the "internal structured clone" algorithm. Transferring objects, now handled by the [StructuredCloneWithTransfer](#) abstract operation, were formerly handled by parts of the algorithm of the `postMessage()` method on the [Window](#) object and the `window/postMessage()` method on the [MessagePort](#) object.

§ 2.9.4. **StructuredClone** (*input*, *targetRealm* [, *memory*])

1. If *memory* was not supplied, let *memory* be an empty map.
2. If *memory* contains an entry with key *input*, then return that entry's value.
3. If [Type](#)(*input*) is Undefined, Null, Boolean, String, or Number, then return *input*.
4. If [Type](#)(*input*) is Symbol, then throw a "[DataCloneError](#)" [DOMException](#).
5. Let *deepClone* be false.
6. If *input* has a *[[BooleanData]]* internal slot, then let *output* be a new Boolean object in *targetRealm* whose *[[BooleanData]]* internal slot value is the *[[BooleanData]]* internal slot value of *input*.
7. Otherwise, if *input* has a *[[NumberData]]* internal slot, then let *output* be a new Number object in *targetRealm* whose *[[NumberData]]* internal slot value is the *[[NumberData]]* internal slot value of *input*.
8. Otherwise, if *input* has a *[[StringData]]* internal slot, then let *output* be a new String object in *targetRealm* whose *[[StringData]]* internal slot value is the *[[StringData]]* internal slot value of *input*.
9. Otherwise, if *input* has a *[[DateValue]]* internal slot, then let *output* be a new Date object in *targetRealm* whose *[[DateValue]]* internal slot value is the *[[DateValue]]* internal slot value of *input*.
10. Otherwise, if *input* has a *[[RegExpMatcher]]* internal slot, then let *output* be a new RegExp object in *targetRealm* whose *[[RegExpMatcher]]* internal slot value is the *[[RegExpMatcher]]* internal slot value of *input*, whose *[[OriginalSource]]* internal slot value is the *[[OriginalSource]]* internal slot value of *input*, and whose *[[OriginalFlags]]* internal slot value is the *[[OriginalFlags]]* internal slot value of *input*.
11. Otherwise, if *input* has an *[[ArrayBufferData]]* internal slot, then:
 1. If [IsDetachedBuffer](#)(*input*) is true, then throw a "[DataCloneError](#)" [DOMException](#).
 2. Let *outputArrayBuffer* be the [%ArrayBuffer%](#) intrinsic object in *targetRealm*.
 3. Let *output* be ? [CloneArrayBuffer](#)(*input*, 0, *outputArrayBuffer*).
12. Otherwise, if *input* has a *[[ViewedArrayBuffer]]* internal slot, then:
 1. Let *buffer* be the value of *input*'s *[[ViewedArrayBuffer]]* internal slot.
 2. Let *bufferClone* be ? [StructuredClone](#)(*buffer*, *targetRealm*, *memory*)}.
 3. If *input* has a *[[DataView]]* internal slot, then let *output* be a new DataView object in *targetRealm* whose *[[DataView]]* internal slot value is true, whose *[[ViewedArrayBuffer]]* internal slot value is *bufferClone*, whose *[[ByteLength]]* internal slot value is the *[[ByteLength]]* internal slot value of *input*, and whose *[[ByteOffset]]* internal slot value is the *[[ByteOffset]]* internal slot value of *input*.
 4. Otherwise:
 1. Assert: *input* has a *[[TypedArrayName]]* internal slot.
 2. Let *constructor* be the intrinsic object listed in column one of [The TypedArray Constructors](#) table for the

- value of *input*'s `[[TypedArrayName]]` internal slot in *targetRealm*.
3. Let *byteOffset* be *input*'s `[[ByteOffset]]` internal slot value.
 4. Let *length* be *input*'s `[[ArrayLength]]` internal slot value.
 5. Let *output* be ? [TypedArrayCreate](#)(*constructor*, « *bufferClone*, *byteOffset*, *length* »).
13. Otherwise, if *input* has `[[MapData]]` internal slot, then:
1. Let *output* be a new Map object in *targetRealm* whose `[[MapData]]` internal slot value is a new empty [List](#).
 2. Set *deepClone* to true.
14. Otherwise, if *input* has `[[SetData]]` internal slot, then:
1. Let *output* be a new Set object in *targetRealm* whose `[[SetData]]` internal slot value is a new empty [List](#).
 2. Set *deepClone* to true.
15. Otherwise, if *input* is an Array exotic object, then:
1. Let *inputLen* be [OrdinaryGetOwnProperty](#)(*input*, "length").`[[value]]`.
 2. Let *outputProto* be the `%ArrayPrototype%` intrinsic object in *targetRealm*.
 3. Let *output* be ! [ArrayCreate](#)(*inputLen*, *outputProto*).
 4. Set *deepClone* to true.
16. Otherwise, if *input* has a `[[Clone]]` internal method, then let *output* be ? *input*.`[[Clone]]`(*targetRealm*, *memory*).
17. Otherwise, if `IsCallable(input)` is true, then throw a "[DataCloneError](#)" [DOMException](#).
18. Otherwise, if *input* has any internal slot other than `[[Prototype]]` or `[[Extensible]]`, then throw a "[DataCloneError](#)" [DOMException](#).
- EXAMPLE 57**
For instance, a `[[PromiseState]]` or `[[WeakMapData]]` internal slot.
19. Otherwise, if *input* is an exotic object, then throw a "[DataCloneError](#)" [DOMException](#).
- EXAMPLE 58**
For instance, a proxy object.
20. Otherwise:
1. Let *output* be a new Object in *targetRealm*.
 2. Set *deepClone* to true.
21. Create an entry in *memory* whose key is *input* and value is *output*.
22. If *deepClone* is true, then:
1. If *input* has a `[[MapData]]` internal slot, then:
 1. Let *inputList* be the value of *input*'s `[[MapData]]` internal slot.
 2. Let *copiedList* be a new empty [List](#).
 3. Repeat for each [Record](#) { `[[key]]`, `[[value]]` } *entry* that is an element of *inputList*,
 1. Let *copiedEntry* be a new [Record](#) { `[[key]]`: *entry*.`[[key]]`, `[[value]]`: *entry*.`[[value]]` }.
 2. If *copiedEntry*.`[[key]]` is not empty, append *copiedEntry* as the last element of *copiedList*.
 4. Let *outputList* be the value of *output*'s `[[MapData]]` internal slot.

5. For each [Record](#) { [[key]], [[value]] } *entry* that is an element of *copiedList*,
 1. Let *outputKey* be ? [StructuredClone](#)(*entry*.[[key]], *targetRealm*, *memory*).
 2. Let *outputValue* be ? [StructuredClone](#)(*entry*.[[value]], *targetRealm*, *memory*).
 3. Add { [[key]]: *outputKey*, [[value]]: *outputValue* } as the last element of *outputList*.
 2. Otherwise, if *input* has a [[SetData]] internal slot, then:
 1. Let *copiedList* be a copy of the value of *input*'s [[SetData]] internal slot.
 2. Let *outputList* be the value of *output*'s [[SetData]] internal slot.
 3. For each *entry* that is an element of *copiedList* that is not empty,
 1. Let *outputEntry* be ? [StructuredClone](#)(*entry*, *targetRealm*, *memory*).
 2. Add *outputEntry* as the last element of *outputList*.
 3. Otherwise:
 1. Let *enumerableKeys* be a new empty [List](#).
 2. For each *key* in ! *input*.[[OwnPropertyKeys]]():
 1. If [Type](#)(*key*) is String, then:
 1. Let *inputDesc* be ! *input*.[[GetOwnProperty]](*key*).
 2. If *inputDesc*.[[Enumerable]] is true, then add *key* as the last element of *enumerableKeys*.
 3. For each *key* in *enumerableKeys*:
 1. If ! [HasOwnProperty](#)(*input*, *key*) is true, then:
 1. Let *inputValue* be ? *input*.[[Get]](*key*, *input*).
 2. Let *outputValue* be ? [StructuredClone](#)(*inputValue*, *targetRealm*, *memory*).
 3. Perform ? [CreateDataProperty](#)(*output*, *key*, *outputValue*).
23. Return *output*.

NOTE:

In general implementations will need to use some kind of serialization and marshalling to implement the creation of objects in *targetRealm*, as *targetRealm* could be in a different [event loop](#) and not easily accessible to the code that invokes [StructuredCloneWithTransfer](#) or [StructuredClone](#).

§ 2.9.5. IsTransferable (*O*)

1. Assert: [Type](#)(*O*) is Object.
2. If *O* has an [[ArrayBufferData]] internal slot, then:
 1. If [IsDetachedBuffer](#)(*O*) is true, then return false.
 2. Return true.
3. Otherwise, if *O* has a [[Detached]] internal slot, then:
 1. If *O*'s [[Detached]] internal slot value is true, then return false.
 2. Return true.
4. Return false.

§ 2.9.6. TransferHelper (*input*, *targetRealm*)

1. If *input* has an `[[ArrayBufferData]]` internal slot, then:
 1. Let *output* be a new `ArrayBuffer` object in *targetRealm* whose `[[ArrayBufferByteLength]]` internal slot value is the `[[ArrayBufferByteLength]]` internal slot value of *input*, and whose `[[ArrayBufferData]]` internal slot value is the `[[ArrayBufferData]]` internal slot value of *input*.
 2. Perform ! `DetachArrayBuffer(input)`.
 3. Return *output*.
2. Return ? *input*.`[[Transfer]]`(*targetRealm*).

§ 3. Semantics, structure, and APIs of HTML documents

§ 3.1. Documents

Every XML and HTML document in an HTML user agent is represented by a `Document` object. [\[DOM\]](#)

The document's address is the *URL associated with a `Document`* (as defined in the DOM standard). It is initially set when the `Document` is created, but that can change during the lifetime of the `Document`; for example, it changes when the user [navigates](#) to a [fragment identifier](#) on the page and when the `pushState()` method is called with a new [URL](#). [\[DOM\]](#)

⚠Warning! Interactive user agents typically expose [the document's address](#) in their user interface. This is the primary mechanism by which a user can tell if a site is attempting to impersonate another.

When a `Document` is created by a [script](#) using the `createDocument()` or `createHTMLDocument()` APIs, [the document's address](#) is the same as [the document's address](#) of the [responsible document](#) specified by the script's [settings object](#), and the `Document` is both [ready for post-load tasks](#) and [completely loaded](#) immediately.

The document's referrer is an [absolute URL](#) that can be set when the `Document` is created. If it is not explicitly set, then its value is the empty string.

Each `Document` object has a **reload override flag** that is originally unset. The flag is set by the `document.open()` and `document.write()` methods in certain situations. When the flag is set, the `Document` also has a **reload override buffer** which is a Unicode string that is used as the source of the document when it is reloaded.

When the user agent is to perform **an overridden reload**, given a [source browsing context](#), it must act as follows:

1. Let *source* be the value of the [browsing context](#)'s [active document](#)'s [reload override buffer](#).
2. Let *address* be the [browsing context](#)'s [active document](#)'s [URL](#).
3. Let *HTTPS state* be the [HTTPS state](#) of the [browsing context](#)'s [active document](#).
4. Let *CSP list* be the [CSP list](#) of the [browsing context](#)'s [active document](#).
5. [Navigate](#) the [browsing context](#) to a new [response](#) whose *body* is *source*, *CSP list* is *CSP list* and *HTTPS state* is *HTTPS state*, with the [exceptions enabled flag](#) and [replacement enabled](#). The [source browsing context](#) is that given to the [overridden reload](#) algorithm. When the [navigate](#) algorithm creates a `Document` object for this purpose, set that `Document`'s [reload override flag](#) and set its [reload override buffer](#) to *source*. Rethrow any exceptions.

When it comes time to [set the document's address](#) in the [navigation algorithm](#), use *address* as the [override URL](#).

§ 3.1.1. The Document object

The DOM specification defines a `Document` interface, which this specification extends significantly:

```

enum DocumentReadyState { "loading", "interactive", "complete" };

[OverrideBuiltins]
partial /*sealed*/ interface Document {
  // resource metadata management
  [PutForwards=href, Unforgeable] readonly attribute Location? location;
  attribute DOMString domain;
  readonly attribute DOMString referrer;
  attribute DOMString cookie;
  readonly attribute DOMString lastModified;
  readonly attribute DocumentReadyState readyState;

  // DOM tree accessors
  getter object (DOMString name);
  attribute DOMString title;
  attribute DOMString dir;
  attribute HTMLElement? body;
  readonly attribute HTMLHeadElement? head;
  [SameObject] readonly attribute HTMLCollection images;
  [SameObject] readonly attribute HTMLCollection embeds;
  [SameObject] readonly attribute HTMLCollection plugins;
  [SameObject] readonly attribute HTMLCollection links;
  [SameObject] readonly attribute HTMLCollection forms;
  [SameObject] readonly attribute HTMLCollection scripts;
  NodeList getElementsByName(DOMString elementName);
  readonly attribute HTMLScriptElement? currentScript;

  // dynamic markup insertion
  Document open(optional DOMString type = "text/html", optional DOMString replace = "");
  WindowProxy open(DOMString url, DOMString name, DOMString features, optional boolean replace =
false);
  void close();
  void write(DOMString... text);
  void writeln(DOMString... text);

  // user interaction
  readonly attribute WindowProxy? defaultView;
  readonly attribute Element? activeElement;
  boolean hasFocus();
  attribute DOMString designMode;
  boolean execCommand(DOMString commandId, optional boolean showUI = false, optional DOMString
value = "");
  boolean queryCommandEnabled(DOMString commandId);
  boolean queryCommandIndeterm(DOMString commandId);
  boolean queryCommandState(DOMString commandId);
  boolean queryCommandSupported(DOMString commandId);
  DOMString queryCommandValue(DOMString commandId);

  // special event handler IDL attributes that only apply to Document objects
  [LenientThis] attribute EventHandler onreadystatechange;
};
Document implements GlobalEventHandlers;
Document implements DocumentAndElementEventHandlers;

```

The [Document](#) has an **HTTPS state** (an [HTTPS state value](#)), initially "none", which represents the security properties of the network channel used to deliver the [Document](#)'s data.

The [Document](#) has a **CSP list**, which is a list of [Content Security Policy](#) objects active in this context. The list is empty unless otherwise specified.

§ 3.1.2. Resource metadata management

This definition is non-normative. Implementation requirements are given below this definition.

`document.referrer`

Returns [the address](#) of the [Document](#) from which the user navigated to this one, unless it was blocked or there was no such document, in which case it returns the empty string.

The `noreferrer` link type can be used to block the referrer.

The **referrer** attribute must return [the document's referrer](#).



This definition is non-normative. Implementation requirements are given below this definition.

`document.cookie [ = value ]`

Returns the HTTP cookies that apply to the [Document](#). If there are no cookies or cookies can't be applied to this resource, the empty string will be returned.

Can be set, to add a new cookie to the element's set of HTTP cookies.

If the contents are [sandboxed into a unique origin](#) (e.g., in an `iframe` with the [sandbox](#) attribute), a `"SecurityError"` [DOMException](#) will be thrown on getting and setting.

The **cookie** attribute represents the cookies of the resource identified by [the document's address](#).

A [Document](#) object that falls into one of the following conditions is a **cookie-averse Document object**:

- A [Document](#) that has no [browsing context](#).
- A [Document](#) whose [address](#) does not use a server-based naming authority.

On getting, if the document is a [cookie-averse Document object](#), then the user agent must return the empty string. Otherwise, if the [Document's origin](#) is an [opaque origin](#), the user agent must throw a `"SecurityError"` [DOMException](#). Otherwise, the user agent must return the [cookie-string](#) for [the document's address](#) for a "non-HTTP" API, decoded using [UTF-8 decoding without BOM](#). [\[COOKIES\]](#) 

On setting, if the document is a [cookie-averse Document object](#), then the user agent must do nothing. Otherwise, if the [Document's origin](#) is an [opaque origin](#), the user agent must throw a `"SecurityError"` [DOMException](#). Otherwise, the user agent must act as it would when [receiving a set-cookie-string](#) for [the document's address](#) via a "non-HTTP" API, consisting of the new value [encoded as UTF-8](#). [\[COOKIES\]](#) [\[ENCODING\]](#)

NOTE:

Since the `cookie` attribute is accessible across frames, the path restrictions on cookies are only a tool to help manage which cookies are sent to which parts of the site, and are not in any way a security feature.

⚠Warning! The `cookie` attribute's getter and setter synchronously access shared state. Since there is no locking mechanism, other browsing contexts in a multiprocess user agent can modify cookies while scripts are running. A site could, for instance, try to read a cookie, increment its value, then write it back out, using the new value of the cookie as a unique identifier for the session; if the site does this twice in two different browser windows at the same time, it might end up using the same "unique" identifier for both sessions, with potentially disastrous effects.



This definition is non-normative. Implementation requirements are given below this definition.

`document.lastModified`

Returns the date of the last modification to the document, as reported by the server, in the form

"MM/DD/YYYY hh:mm:ss", in the user's local time zone.

If the last modification date is not known, the current time is returned instead.

The **lastModified** attribute, on getting, must return the date and time of the [Document](#)'s source file's last modification, in the user's local time zone, in the following format:

1. The month component of the date.
2. A U+002F SOLIDUS character (/).
3. The day component of the date.
4. A U+002F SOLIDUS character (/).
5. The year component of the date.
6. A U+0020 SPACE character.
7. The hours component of the time.
8. A U+003A COLON character (:).
9. The minutes component of the time.
10. A U+003A COLON character (:).
11. The seconds component of the time.

All the numeric components above, other than the year, must be given as two [ASCII digits](#) representing the number in base ten, zero-padded if necessary. The year must be given as the shortest possible string of four or more [ASCII digits](#) representing the number in base ten, zero-padded if necessary.

The [Document](#)'s source file's last modification date and time must be derived from relevant features of the networking protocols used, e.g., from the value of the HTTP Last-Modified header of the document, or from metadata in the file system for local files. If the last modification date and time are not known, the attribute must return the current date and time in the above format.



This definition is non-normative. Implementation requirements are given below this definition.

***document*.readyState**

Returns "loading" while the [Document](#) is loading, "interactive" once it is finished parsing but still loading sub-resources, and "complete" once it has loaded.

Thereadystatechange event fires on the [Document](#) object when this value changes.

Each document has a **current document readiness**. When a [Document](#) object is created, it must have its [current document readiness](#) set to the string "loading" if the document is associated with an [HTML parser](#), an [XML parser](#), or an XSLT processor, and to the string "complete" otherwise. Various algorithms during page loading affect this value. When the value is set, the user agent must [fire a simple event](#) named readystatechange at the [Document](#) object.

A [Document](#) is said to have an **active parser** if it is associated with an [HTML parser](#) or an [XML parser](#) that has not yet been [stopped](#) or

[aborted](#).

The **readyState** IDL attribute must, on getting, return the [current document readiness](#).

§ 3.1.3. DOM tree accessors

The [<html>](#) element of a document is the document's root element, if there is one and it's an [<html>](#) element, or null otherwise.



This definition is non-normative. Implementation requirements are given below this definition.

`document . head`

Returns the [<head>](#) element.

The [<head>](#) element of a document is the first [<head>](#) element that is a child of the [<html>](#) element, if there is one, or null otherwise.

The **head** attribute, on getting, must return the [<head>](#) element of the document (a [<head>](#) element or null).



This definition is non-normative. Implementation requirements are given below this definition.

`document . title [ = value ]`

Returns the document's title, as given by the [<title>](#) element for HTML and as given by the SVG [<title>](#) element for SVG.

Can be set, to update the document's title. If there is no appropriate element to update, the new value is ignored.

The [<title>](#) element of a document is the first [<title>](#) element in the document (in [tree order](#)), if there is one, or null otherwise.

The **title** attribute must, on getting, run the following algorithm:

1. If the [root element](#) is an [<svg>](#) element in the [SVG namespace](#), then let *value* be a concatenation of the data of all the child Text nodes of the first [<title>](#) element in the [SVG namespace](#) that is a child of the [root element](#). [\[SVG11\]](#)
2. Otherwise, let *value* be a concatenation of the data of all the child Text nodes of the [<title>](#) element, in [tree order](#), or the empty string if the [<title>](#) element is null.
3. [Strip and collapse whitespace](#) in *value*.
4. Return *value*.

On setting, the steps corresponding to the first matching condition in the following list must be run:

↪ If the [root element](#) is an [<svg>](#) element in the [SVG namespace](#) [\[SVG11\]](#)

1. Let *element* be the first [<title>](#) element in the [SVG namespace](#) that is a child of the [root element](#), if any. If there isn't one, create a [<title>](#) element in the [SVG namespace](#), insert it as the first child of the [root element](#), and let *element* be that element. [\[SVG11\]](#)
2. Act as if the [textContent](#) IDL attribute of *element* was set to the new value being assigned.

↪ If the [root element](#) is in the [HTML namespace](#)

1. If the [<title>](#) element is null and the [<head>](#) element is null, then abort these steps.
2. If the [<title>](#) element is null, then create a new **title** element and [append](#) it to the [<head>](#) element, and let *element* be the newly created element; otherwise, let *element* be the [<title>](#) element.
3. Act as if the [textContent](#) IDL attribute of *element* was set to the new value being assigned.

↪ Otherwise

Do nothing.



This definition is non-normative. Implementation requirements are given below this definition.

`document . body [ = value ]`

Returns the [<body>](#) element.

Can be set, to replace the [<body>](#) element.

If the new value is not a body or [<frameset>](#) element, this will throw a `HierarchyRequestError` exception.

The [<body>](#) element of a document is the first child of the [<html>](#) element that is either a [<body>](#) element or a [<frameset>](#) element. If there is no such element, it is null.

The **body** attribute, on getting, must return the [<body>](#) element of the document (either a [<body>](#) element, a [<frameset>](#) element, or null). On setting, the following algorithm must be run:

1. If the new value is not a [<body>](#) or [<frameset>](#) element, then throw a `HierarchyRequestError` exception and abort these steps.
2. Otherwise, if the new value is the same as the [<body>](#) element, do nothing. Abort these steps.
3. Otherwise, if the [<body>](#) element is not null, then replace that element with the new value in the DOM, as if the root element's `replaceChild()` method had been called with the new value and the incumbent [<body>](#) element as its two arguments respectively, then abort these steps.
4. Otherwise, if there is no root element, throw a `HierarchyRequestError` exception and abort these steps.
5. Otherwise, the [<body>](#) element is null, but there's a root element. Append the new value to the root element.



This definition is non-normative. Implementation requirements are given below this definition.

`document . images`

Returns an `HTMLCollection` of the [](#) elements in the [Document](#).

`document . embeds`

`document . plugins`

Return an `HTMLCollection` of the [<embed>](#) elements in the [Document](#).

`document . links`

Returns an `HTMLCollection` of the [<a>](#) and [<area>](#) elements in the [Document](#) that have [href](#) attributes.

`document . forms`

Return an `HTMLCollection` of the [<form>](#) elements in the [Document](#).

`document . scripts`

Return an `HTMLCollection` of the [<script>](#) elements in the [Document](#).

The **images** attribute must return an `HTMLCollection` rooted at the [Document](#) node, whose filter matches only [](#) elements.

The **embeds** attribute must return an `HTMLCollection` rooted at the [Document](#) node, whose filter matches only [<embed>](#) elements.

The **plugins** attribute must return the same object as that returned by the **embeds** attribute.

The **links** attribute must return an `HTMLCollection` rooted at the [Document](#) node, whose filter matches only [<a>](#) elements with [href](#) attributes and [<area>](#) elements with [href](#) attributes.

The **forms** attribute must return an `HTMLCollection` rooted at the [Document](#) node, whose filter matches only [<form>](#) elements.

The **scripts** attribute must return an `HTMLCollection` rooted at the [Document](#) node, whose filter matches only [<script>](#)

elements.



This definition is non-normative. Implementation requirements are given below this definition.

`collection = document . getElementsByName( name )`

Returns a `NodeList` of elements in the [Document](#) that have a `name` attribute with the value `name`.

The '[getElementsByName\(name \)](#)' method takes a string `name`, and must return a [live](#) `NodeList` containing all the [html elements](#) in that document that have a `name` attribute whose value is equal to the `name` argument (in a [case-sensitive](#) manner), in [tree order](#). When the method is invoked on a [Document](#) object again with the same argument, the user agent may return the same as the object returned by the earlier call. In other cases, a new `NodeList` object must be returned.



This definition is non-normative. Implementation requirements are given below this definition.

`document . currentScript`

Returns the [<script>](#) element that is currently executing. In the case of reentrant script execution, returns the one that most recently started executing amongst those that have not yet finished executing.

Returns null if the [Document](#) is not currently executing a script element (e.g., because the running script is an event handler, or a timeout).

The **`currentScript`** attribute, on getting, must return the value to which it was most recently initialized. When the [Document](#) is created, the `currentScript` must be initialized to null.



The [Document](#) interface [supports named properties](#). The [supported property names](#) at any moment consist of the values of the `name` content attributes of all the [<applet>](#), [exposed <embed>](#), [<form>](#), [<iframe>](#), [](#), and [exposed <object>](#) elements in the [Document](#) that have non-empty `name` content attributes, and the values of the `id` content attributes of all the [applet](#) and [exposed <object>](#) elements in the [Document](#) that have non-empty `id` content attributes, and the values of the `id` content attributes of all the [](#) elements in the [Document](#) that have both non-empty `name` content attributes and non-empty `id` content attributes. The [supported property names](#) must be in [tree order](#), ignoring later duplicates, with values from `id` attributes coming before values from `name` attributes when the same element contributes both.

To [determine the value of a named property](#) `name` when the [Document](#) object is indexed for property retrieval, the user agent must return the value obtained using the following steps:

1. Let `elements` be the list of [named elements](#) with the name `name` in the [Document](#).

NOTE:

There will be at least one such element, by definition.

2. If `elements` has only one element, and that element is an [iframe](#) element, then return the [WindowProxy](#) object of the [nested browsing context](#) represented by that [<iframe>](#) element, and abort these steps.
3. Otherwise, if `elements` has only one element, return that element and abort these steps.
4. Otherwise return an [HTMLCollection](#) rooted at the [Document](#) node, whose filter matches only [named elements](#) with the name `name`.

Named elements with the name `name`, for the purposes of the above algorithm, are those that are either:

- [<applet>](#), [exposed <embed>](#), [<form>](#), [<iframe>](#), [](#), or [exposed <object>](#) elements that have a `name` content attribute

whose value is `name`, or

- `applet` or `exposed <object>` elements that have an `id` content attribute whose value is `name`, or
- `<img>` elements that have an `id` content attribute whose value is `name`, and that have a non-empty `name` content attribute present also.

An `embed` or `<object>` element is said to be **exposed** if it has no `exposed` object ancestor, and, for `<object>` elements, is additionally either not showing its `fallback content` or has no object or `embed` descendants.



NOTE:

The `dir` attribute on the `Document` interface is defined along with the `dir` content attribute.

§ 3.1.4. Loading XML documents

```
partial interface XMLDocument {
  boolean load(DOMString url);
};
```

The `'load(url)'` method must run the following steps:

1. Let *document* be the `XMLDocument` object on which the method was invoked.
2. **Parse** *url*, relative to the `entry settings object`. If this is not successful, throw a `"SyntaxError" DOMException` and abort these steps. Otherwise, let *urlRecord* be the `resulting URL record`.
3. If *urlRecord*'s `origin` is not the same as the `origin` of *document*, throw a `"SecurityError" DOMException` and abort these steps.
4. Remove all child nodes of *document*, without firing any mutation events.
5. Set the `current document readiness` of *document* to "loading".
6. Run the remainder of these steps **in parallel**, and return true from the method.
7. Let *result* be a `Document` object.
8. Let *success* be false.
9. Let *request* be a new `request` whose `URL` is *urlRecord*, `client` is `entry settings object`, `destination` is "subresource", `synchronous flag` is set, `mode` is "same-origin", `credentials mode` is "same-origin", and whose `use-URL-credentials flag` is set.
10. Let *response* be the result of **fetching** *request*.
11. If *response*'s `Content-Type metadata` is an `XML MIME type`, then run these substeps:
 1. Create a new `XML parser` associated with the *result* document.
 2. Pass this parser *response*'s `body`.
 3. If there is an XML well-formedness or XML namespace well-formedness error, then remove all child nodes from *result*. Otherwise let *success* be true.
12. Queue a task to run the following steps.
 1. Set the `current document readiness` of *document* to "complete".
 2. Replace all the children of *document* by the children of *result* (even if it has no children), firing mutation events as if a `DocumentFragment` containing the new children had been inserted.

3. [Fire a simple event](#) named `load` at `document`.

§ 3.2. Elements

§ 3.2.1. Semantics

Elements, attributes, and attribute values in HTML are defined (by this specification) to have certain meanings (semantics). For example, the `<ol>` element represents an ordered list, and the `lang` attribute represents the language of the content.

These definitions allow HTML processors, like web browsers and search engines, to present documents and applications consistently in different contexts.

EXAMPLE 59

In this example the HTML headings may be presented as large text in a desktop browser, or standard size text in bold in a mobile browser. In both cases the semantic information remains the same - that the `<h1>` and `<h2>` elements represent headings.

```
<!doctype html>
<html lang="en">
  <head>
    <title>Favorite books</title>
  </head>
  <body>
    <header>
      
    </header>
    <main>
      <h1>Favorite books</h1>
      <p>These are a few of my favorite books.</p>
      <h2>The Belgariad</h2>
      <p>Five books by David and Leigh Eddings.</p>
      <h2>The Hitchhiker's Guide to the Galaxy</h2>
      <p>A trilogy of five books by Douglas Adams.</p>
    </main>
  </body>
</html>
```

This semantic information is critical to assistive technologies. For example, a screen reader will query the browser for semantic information and use that information to present the document or application in synthetic speech.

In some cases assistive technologies use semantic information to provide additional functionality. A speech recognition tool might provide a voice command for moving focus to the start of the `<main>` element for example.

When the appropriate HTML element or attribute is not used, it deprives HTML processors of valuable semantic information.

EXAMPLE 60

In this example styling may be used to create a visual representation of headings and other components, but because the appropriate HTML elements have not been used there is little semantic information available to web browsers, search engines and assistive technologies.

```
<!doctype html>
<html lang="en">
  <head>
    <title>Favorite books</title>
  </head>
  <body>
    <div class="header">
      
    </div>
    <div class="main">
      <span class="largeHeading">Favorite books</span>
      <p>These are a few of my favorite books.</p>
      <span class="smallHeading">The Belgariad</span>
      <p>Five books by David and Leigh Eddings.</p>
      <span class="smallHeading">The Hitchhiker's Guide to the Galaxy</span>
      <p>A trilogy of five books by Douglas Adams.</p>
    </div>
  </body>
</html>
```

A document can change dynamically while it is being processed. Scripting and other mechanisms can be used to change attribute values, text, or the entire document structure. The semantics of a document are therefore based on the document's state at a particular instance in time, but may also change in response to external events. User agents must update their presentation of the document to reflect these changes.

EXAMPLE 61

In this example the `<audio>` element is used to play a music track. The **controls** attribute is used to show the user agent player, and as the music plays the controls are updated to indicate progress. The available semantic information is updated in response to these changes.

```
<audio src="comfortablynumb.mp3" controls>
```

§ **3.2.2. Elements in the DOM**

The nodes representing [html elements](#) in the DOM must implement, and expose to scripts, the interfaces listed for them in the relevant sections of this specification. This includes [html elements](#) in [XML documents](#), even when those documents are in another context (e.g., inside an XSLT transform).

Elements in the DOM **represent** things; that is, they have intrinsic *meaning*, also known as semantics.

EXAMPLE 62

For example, an `<ol>` element represents an ordered list.

The basic interface, from which all the [html elements'](#) interfaces inherit, and which must be used by elements that have no additional requirements, is the [HTMLElement](#) interface.

```

interface HTMLElement : Element {
    // metadata attributes
    attribute DOMString title;
    attribute DOMString lang;
    attribute boolean translate;
    attribute DOMString dir;
    [SameObject] readonly attribute DOMStringMap dataset;

    // user interaction
    attribute boolean hidden;
    void click();
    attribute long tabIndex;
    void focus();
    void blur();
    attribute DOMString accessKey;
    attribute boolean draggable;
    [PutForwards=value] readonly attribute DOMTokenList dropzone;
    attribute HTMLMenuElement? contextMenu;
    attribute boolean spellcheck;
    void forceSpellCheck();
};
HTMLElement implements GlobalEventHandlers;
HTMLElement implements DocumentAndElementEventHandlers;
HTMLElement implements ElementContentEditable;

interface HTMLUnknownElement : HTMLElement { };

```

The [HTMLElement](#) interface holds methods and attributes related to a number of disparate features, and the members of this interface are therefore described in various different sections of this specification.

The [HTMLUnknownElement](#) interface must be used for [html elements](#) that are not defined by this specification (or [other applicable specifications](#)).

§ 3.2.3. Element definitions

Each element in this specification has a definition that includes the following information:

Categories

A list of [categories](#) to which the element belongs. These are used when defining the [content models](#) for each element.

Contexts in which this element can be used

A *non-normative* description of where the element can be used. This information is redundant with the content models of elements that allow this one as a child, and is provided only as a convenience.

NOTE:

For simplicity, only the most specific expectations are listed. For example, an element that is both [flow content](#) and [phrasing content](#) can be used anywhere that either [flow content](#) or [phrasing content](#) is expected, but since anywhere that [flow content](#) is expected, [phrasing content](#) is also expected (since all [phrasing content](#) is [flow content](#)), only "where [phrasing content](#) is expected" will be listed.

Content model

A normative description of what content must be included as children and descendants of the element.

Tag omission in text/html

A *non-normative* description of whether, in the [text/html](#) syntax, the [start](#) and [end](#) tags can be omitted. This information is redundant with the normative requirements given in the [optional tags](#) section, and is provided in the element definitions only as a convenience.

Content attributes

A normative list of attributes that may be specified on the element (except where otherwise disallowed), along with

non-normative descriptions of those attributes. (The content to the left of the dash is normative, the content to the right of the dash is not.)

Allowed [ARIA role attribute](#) values

A normative list of [ARIA role attribute](#) values that may be specified on the element (except where otherwise disallowed). Each value is linked to a non normative description.

Allowed [ARIA state and property attributes](#)

Links to the [Global aria-* attributes](#) list and the [allowed roles, states and properties](#) table.

DOM interface

A normative definition of a DOM interface that such elements must implement.

This is then followed by a description of what the element [represents](#), along with any additional normative conformance criteria that may apply to authors and implementations. Examples are sometimes also included.

§ 3.2.3.1. *Attributes*

Except where otherwise specified, attributes on [html elements](#) may have any string value, including the empty string. Except where explicitly stated, there is no restriction on what text can be specified in such attributes.

§ 3.2.4. **Content models**

Each element defined in this specification has a content model: a description of the element's expected [contents](#). An [HTML element](#) must have contents that match the requirements described in the element's content model. The **contents** of an element are its children in the DOM, except for [<template>](#) elements, where the children are those in the [template contents](#) (a separate DocumentFragment assigned to the element when the element is created).

The [space characters](#) are always allowed between elements. User agents represent these characters between elements in the source markup as Text nodes in the DOM. Empty Text nodes and Text nodes consisting of just sequences of those characters are considered **inter-element whitespace**.

[Inter-element whitespace](#), comment nodes, and processing instruction nodes must be ignored when establishing whether an element's contents match the element's content model or not, and must be ignored when following algorithms that define document and element semantics.

NOTE:

Thus, an element *A* is said to be preceded or followed by a second element *B* if *A* and *B* have the same parent node and there are no other element nodes or Text nodes (other than [inter-element whitespace](#)) between them. Similarly, a node is the only child of an element if that element contains no other nodes other than [inter-element whitespace](#), comment nodes, and processing instruction nodes.

Authors must not use [html elements](#) anywhere except where they are explicitly allowed, as defined for each element, or as explicitly required by other specifications. For XML compound documents, these contexts could be inside elements from other namespaces, if those elements are defined as providing the relevant contexts.

EXAMPLE 63

For example, the Atom specification defines a content element. When its type attribute has the value xhtml, the Atom specification requires that it contain a single HTML [<div>](#) element. Thus, a [<div>](#) element is allowed in that context, even though this is not explicitly normatively stated by this specification. [\[RFC4287\]](#)

In addition, [html elements](#) may be orphan nodes (i.e., without a parent node).

EXAMPLE 64

For example, creating a `<td>` element and storing it in a global variable in a script is conforming, even though `<td>` elements are otherwise only supposed to be used inside `<tr>` elements.

```
var data = {
  name: "Banana",
  cell: document.createElement('td'),
};
```

3.2.4.1. The "nothing" content model

When an element's content model is **nothing**, the element must contain no Text nodes (other than [inter-element whitespace](#)) and no element nodes.

NOTE:

Most HTML elements whose content model is "nothing" are also, for convenience, [void elements](#) (elements that have no [end tag](#) in the [HTML syntax](#)). However, these are entirely separate concepts.

3.2.4.2. Kinds of content

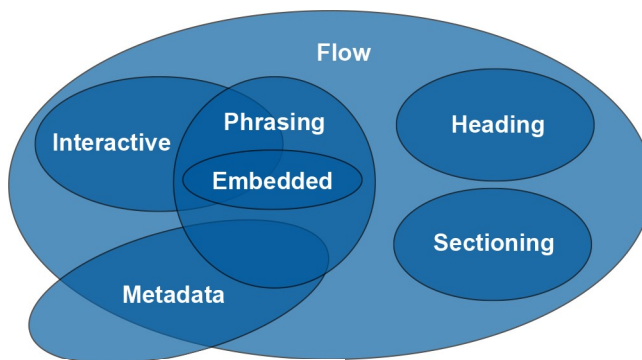
Each element in HTML falls into zero or more **categories** that group elements with similar characteristics together. The following broad categories are used in this specification:

⇒ [§3.2.4.2.1 metadata content](#) , [§3.2.4.2.2 flow content](#) , [§3.2.4.2.3 sectioning content](#) , [§3.2.4.2.4 heading content](#) , [§3.2.4.2.5 phrasing content](#) , [§3.2.4.2.6 embedded content](#) , [§3.2.4.2.7 interactive content](#)

NOTE:

Some elements also fall into other categories, which are defined in other parts of this specification.

These categories are related as follows:



Sectioning content, heading content, phrasing content, embedded content, and interactive content are all types of flow content. Metadata is sometimes flow content. Metadata and interactive content are sometimes phrasing content. Embedded content is also a type of phrasing content, and sometimes is interactive content.

Other categories are also used for specific purposes, e.g., form controls are specified using a number of categories to define common requirements. Some elements have unique requirements and do not fit into any particular category.

§ 3.2.4.2.1. METADATA CONTENT

Metadata content is content that sets up the presentation or behavior of the rest of the content, or that sets up the relationship of the document with other documents, or that conveys other "out of band" information.

⇒ `<base>` , `<link>` , `<meta>` , `<noscript>` , `<script>` , `<style>` , `<template>` , `<title>`

§ 3.2.4.2.2. FLOW CONTENT

Most elements that are used in the body of documents and applications are categorized as **flow content**.

⇒ [<a>](#), [<abbr>](#), [<address>](#), [<area>](#) (if it is a descendant of a [<map>](#) element), [<article>](#), [<aside>](#), [<audio>](#), [](#), [<bdi>](#), [<bdo>](#), [<blockquote>](#), [
](#), [<button>](#), [<canvas>](#), [<cite>](#), [<code>](#), [<data>](#), [<datalist>](#), [<del>](#), [<details>](#), [<dfn>](#), [<div>](#), [<dl>](#), [](#), [<embed>](#), [<fieldset>](#), [<figure>](#), [<footer>](#), [<form>](#), [<h1>](#), [<h2>](#), [<h3>](#), [<h4>](#), [<h5>](#), [<h6>](#), [<header>](#), [<hr>](#), [<i>](#), [<iframe>](#), [](#), [<input>](#), [<ins>](#), [<kbd>](#), [<keygen>](#), [<label>](#), [<main>](#), [<map>](#), [<mark>](#), [<math>](#), [<menu>](#), [<meter>](#), [<nav>](#), [<noscript>](#), [<object>](#), [](#), [<output>](#), [<p>](#), [<picture>](#), [<pre>](#), [<progress>](#), [<q>](#), [<ruby>](#), [<s>](#), [<samp>](#), [<script>](#), [<section>](#), [<select>](#), [<small>](#), [](#), [](#), [<sub>](#), [<sup>](#), [<svg>](#), [<table>](#), [<template>](#), [<textarea>](#), [<time>](#), [<u>](#), [](#), [<var>](#), [<video>](#), [<wbr>](#), [text](#)

§ 3.2.4.2.3. SECTIONING CONTENT

Sectioning content is content that defines the scope of [headings](#) and [<footers>](#).

⇒ [<article>](#), [<aside>](#), [<nav>](#), [<section>](#)

Each [sectioning content](#) element potentially has a heading and an [outline](#). See the section on [§4.3.10 Headings and sections](#) for further details.

NOTE:

There are also certain elements that are [sectioning roots](#). These are distinct from [sectioning content](#), but they can also have an [outline](#).

§ 3.2.4.2.4. HEADING CONTENT

Heading content defines the header of a section (whether explicitly marked up using [sectioning content](#) elements, or implied by the heading content itself).

⇒ [<h1>](#), [<h2>](#), [<h3>](#), [<h4>](#), [<h5>](#), [<h6>](#)

§ 3.2.4.2.5. PHRASING CONTENT

Phrasing content is the text of the document, as well as elements that mark up that text at the intra-paragraph level. Runs of [phrasing content](#) form [paragraphs](#).

⇒ [<a>](#), [<abbr>](#), [<area>](#) (if it is a descendant of a [<map>](#) element), [<audio>](#), [](#), [<bdi>](#), [<bdo>](#), [
](#), [<button>](#), [<canvas>](#), [<cite>](#), [<code>](#), [<data>](#), [<datalist>](#), [<del>](#), [<dfn>](#), [](#), [<embed>](#), [<i>](#), [<iframe>](#), [](#), [<input>](#), [<ins>](#), [<kbd>](#), [<keygen>](#), [<label>](#), [<map>](#), [<mark>](#), [<math>](#), [<meter>](#), [<noscript>](#), [<object>](#), [<output>](#), [<picture>](#), [<progress>](#), [<q>](#), [<ruby>](#), [<s>](#), [<samp>](#), [<script>](#), [<select>](#), [<small>](#), [](#), [](#), [<sub>](#), [<sup>](#), [<svg>](#), [<template>](#), [<textarea>](#), [<time>](#), [<u>](#), [<var>](#), [<video>](#), [<wbr>](#), [text](#)

NOTE:

Most elements that are categorized as phrasing content can only contain elements that are themselves categorized as phrasing content, not any flow content.

Text, in the context of content models, means either nothing, or Text nodes. [Text](#) is sometimes used as a content model on its own, but is also [phrasing content](#), and can be [inter-element whitespace](#) (if the Text nodes are empty or contain just [space characters](#)).

Text nodes and attribute values must consist of [Unicode characters](#), must not contain U+0000 characters, must not contain permanently undefined Unicode characters (noncharacters), and must not contain [control characters](#) other than [space characters](#).

This specification includes extra constraints on the exact value of Text nodes and attribute values depending on their precise context.

For elements in HTML, the constraints of the [Text content model](#) also depends on the [kind of element](#). For instance, an "<" inside a [<textarea>](#) element does not need to be escaped in HTML because [<textarea>](#) is an [escapable raw text](#) element.

(This does not apply to XHTML. In XHTML, the [kind of element](#) doesn't affect the constraints of [content model: Text](#).)

§ 3.2.4.2.6. EMBEDDED CONTENT

Embedded content is content that imports another resource into the document, or content from another vocabulary that is [inserted into the document](#).

⇒ [<audio>](#), [<canvas>](#), [<embed>](#), [<iframe>](#), [](#), [<math>](#), [<object>](#), [<picture>](#), [<svg>](#), [<video>](#)

Elements that are from namespaces other than the [HTML namespace](#) and that convey content but not metadata, are [embedded content](#) for the purposes of the content models defined in this specification. (For example, MathML, or SVG.)

Some embedded content elements can have **fallback content**: content that is to be used when the external resource cannot be used (e.g., because it is of an unsupported format). The element definitions state what the fallback is, if any.

§ 3.2.4.2.7. INTERACTIVE CONTENT

Interactive content is content that is specifically intended for user interaction.

⇒ [<a>](#) (if the [href](#) attribute is present), [<audio>](#) (if the [controls](#) attribute is present), [<button>](#), [<details>](#), [<embed>](#), [<iframe>](#), [](#) (if the [usemap](#) attribute is present), [<input>](#) (if the [type](#) attribute is *not* in the [hidden](#) state), [<keygen>](#), [<label>](#), [<select>](#), [<textarea>](#), [<video>](#) (if the [controls](#) attribute is present)

The [tabindex](#) attribute can also make any element into [interactive content](#).

§ 3.2.4.2.8. PALPABLE CONTENT

As a general rule, elements whose content model allows any [flow content](#) or [phrasing content](#) should have at least one node in its [contents](#) that is [palpable content](#) and that does not have the [hidden](#) attribute specified.

NOTE:

Palpable content makes an element non-empty by providing either some descendant non-empty [text](#), or else something users can hear ([<audio>](#) elements) or view ([<video>](#) or [](#) or [<canvas>](#) elements) or otherwise interact with (for example, interactive form controls).

This requirement is not a hard requirement, however, as there are many cases where an element can be empty legitimately, for example when it is used as a placeholder which will later be filled in by a script, or when the element is part of a template and would on most pages be filled in but on some pages is not relevant.

Conformance checkers are encouraged to provide a mechanism for authors to find elements that fail to fulfill this requirement, as an authoring aid.

The following elements are palpable content:

⇒ [<a>](#), [<abbr>](#), [<address>](#), [<article>](#), [<aside>](#), [<audio>](#) (if the [controls](#) attribute is present), [](#), [<bdi>](#), [<bdo>](#), [<blockquote>](#), [<button>](#), [<canvas>](#), [<cite>](#), [<code>](#), [<data>](#), [<details>](#), [<dfn>](#), [<div>](#), [<dl>](#) (if the element's children include at least one name-value group), [](#), [<embed>](#), [<fieldset>](#), [<figure>](#), [<footer>](#), [<form>](#), [<h1>](#), [<h2>](#), [<h3>](#), [<h4>](#), [<h5>](#), [<h6>](#), [<header>](#), [<i>](#), [<iframe>](#), [](#), [<input>](#) (if the [type](#) attribute is *not* in the [hidden](#) state), [<ins>](#), [<kbd>](#), [<keygen>](#), [<label>](#), [<main>](#), [<map>](#), [<mark>](#), [<math>](#), [<meter>](#), [<nav>](#), [<object>](#), [](#) (if the element's children include at least one [](#) element), [<output>](#), [<p>](#), [<pre>](#), [<progress>](#), [<q>](#), [<ruby>](#), [<s>](#), [<samp>](#), [<section>](#), [<select>](#), [<small>](#), [](#), [](#), [<sub>](#), [<sup>](#), [<svg>](#), [<table>](#), [<textarea>](#), [<time>](#), [<u>](#), [](#) (if the element's children include at least one [](#) element), [<var>](#), [<video>](#), [text](#) that is not [inter-element whitespace](#)

§ 3.2.4.2.9. SCRIPT-SUPPORTING ELEMENTS

Script-supporting elements are those that do not [represent](#) anything themselves (i.e., they are not rendered), but are used to support scripts, e.g., to provide functionality for the user.

The following elements are script-supporting elements:

⇒ [<script>](#), [<template>](#)

3.2.4.3. Transparent content models

Some elements are described as **transparent**; they have "transparent" in the description of their content model. The content model of a [transparent](#) element is derived from the content model of its parent element: the elements required in the part of the content model that is "transparent" are the same elements as required in the part of the content model of the parent of the transparent element in which the transparent element finds itself.

EXAMPLE 65

For instance, an [<ins>](#) element inside a [<ruby>](#) element cannot contain an [<rt>](#) element, because the part of the [<ruby>](#) element's content model that allows [<ins>](#) elements is the part that allows [phrasing content](#), and the [<rt>](#) element is not [phrasing content](#).

NOTE:

In some cases, where transparent elements are nested in each other, the process has to be applied iteratively.

EXAMPLE 66

Consider the following markup fragment:

```
<p><object><param><ins><map><a href="/">Apples</a></map></ins></object></p>
```

To check whether "Apples" is allowed inside the [<a>](#) element, the content models are examined. The [<a>](#) element's content model is transparent, as is the [<map>](#) element's, as is the [<ins>](#) element's, as is the part of the [<object>](#) element's in which the [<ins>](#) element is found. The [<object>](#) element is found in the [<p>](#) element, whose content model is [phrasing content](#). Thus, "Apples" is allowed, as text is [phrasing content](#).

When a transparent element has no parent, then the part of its content model that is "transparent" must instead be treated as accepting any [flow content](#).

3.2.4.4. Paragraphs

NOTE:

The term [paragraph](#) as defined in this section is used for more than just the definition of the [<p>](#) element. The [paragraph](#) concept defined here is used to describe how to interpret documents. The [<p>](#) element is merely one of several ways of marking up a [paragraph](#).

A **paragraph** is typically a run of [phrasing content](#) that forms a block of text with one or more sentences that discuss a particular topic, as in typography, but can also be used for more general thematic grouping. For instance, an address is also a paragraph, as is a part of a form, a byline, or a stanza in a poem.

EXAMPLE 67

In the following example, there are two paragraphs in a section. There is also a heading, which contains phrasing content that is not a paragraph. Note how the comments and [inter-element whitespace](#) do not form paragraphs.

```
<section>
  <h2>Example of paragraphs</h2>
  This is the <em>first</em> paragraph in this example.
  <p>This is the second.</p>
  <!-- This is not a paragraph. -->
</section>
```

Paragraphs in [flow content](#) are defined relative to what the document looks like without the [<a>](#), [<ins>](#), [<del>](#), and [<map>](#) elements complicating matters, since those elements, with their hybrid content models, can straddle paragraph boundaries, as shown in the first two examples below.

NOTE:

Generally, having elements straddle paragraph boundaries is best avoided. Maintaining such markup can be difficult.

EXAMPLE 68

The following example takes the markup from the earlier example and puts `<ins>` and `<del>` elements around some of the markup to show that the text was changed (though in this case, the changes admittedly don't make much sense). Notice how this example has exactly the same paragraphs as the previous one, despite the `<ins>` and `<del>` elements — the `<ins>` element straddles the heading and the first paragraph, and the `<del>` element straddles the boundary between the two paragraphs.

```
<section>
  <ins><h1>Example of paragraphs</h1>
  This is the <em>first</em> paragraph in</ins> this example<del>.
  <p>This is the second.</p></del>
  <!-- This is not a paragraph. -->
</section>
```

Let *view* be a view of the DOM that replaces all `<a>`, `<ins>`, `<del>`, and `<map>` elements in the document with their [contents](#). Then, in *view*, for each run of sibling [phrasing content](#) nodes uninterrupted by other types of content, in an element that accepts content other than [phrasing content](#) as well as [phrasing content](#), let *first* be the first node of the run, and let *last* be the last node of the run. For each such run that consists of at least one node that is neither [embedded content](#) nor [inter-element whitespace](#), a paragraph exists in the original DOM from immediately before *first* to immediately after *last*. (Paragraphs can thus span across `<a>`, `<ins>`, `<del>`, and `<map>` elements.)

Conformance checkers may warn authors of cases where they have paragraphs that overlap each other (this can happen with `<object>`, `<video>`, `<audio>`, and `<canvas>` elements, and indirectly through elements in other namespaces that allow HTML to be further embedded therein, like `<svg>` or `<math>`).

A [paragraph](#) is also formed explicitly by `<p>` elements.

NOTE:

The `<p>` element can be used to wrap individual paragraphs when there would otherwise not be any content other than phrasing content to separate the paragraphs from each other.

EXAMPLE 69

In the following example, the link spans half of the first paragraph, all of the heading separating the two paragraphs, and half of the second paragraph. It straddles the paragraphs and the heading.

```
<header>
  Welcome!
  <a href="about.html">
    This is home of...
    <h1>The Falcons!</h1>
    The Lockheed Martin multirole jet fighter aircraft!
  </a>
  This page discusses the F-16 Fighting Falcon's innermost secrets.
</header>
```

Here is another way of marking this up, this time showing the paragraphs explicitly, and splitting the one link element into three:

```
<header>
  <p>Welcome! <a href="about.html">This is home of...</a></p>
  <h1><a href="about.html">The Falcons!</a></h1>
  <p><a href="about.html">The Lockheed Martin multirole jet
    fighter aircraft!</a> This page discusses the F-16 Fighting
    Falcon's innermost secrets.</p>
</header>
```

EXAMPLE 70

It is possible for paragraphs to overlap when using certain elements that define fallback content. For example, in the following section:

```
<section>
  <h2>My Cats</h2>
  You can play with my cat simulator.
  <object data="cats.sim">
    To see the cat simulator, use one of the following links:
    <ul>
      <li><a href="cats.sim">Download simulator file</a>
      <li><a href="https://sims.example.com/watch?v=LYds5xY4INU">Use online simulator</a>
    </ul>
    Alternatively, upgrade to the Mellblom Browser.
  </object>
  I'm quite proud of it.
</section>
```

There are five paragraphs:

1. The paragraph that says "You can play with my cat simulator. *object* I'm quite proud of it.", where *object* is the `<object>` element.
2. The paragraph that says "To see the cat simulator, use one of the following links:".
3. The paragraph that says "Download simulator file".
4. The paragraph that says "Use online simulator".
5. The paragraph that says "Alternatively, upgrade to the Mellblom Browser.".

The first paragraph is overlapped by the other four. A user agent that supports the "cats.sim" resource will only show the first one, but a user agent that shows the fallback will confusingly show the first sentence of the first paragraph as if it was in the same paragraph as the second one, and will show the last paragraph as if it was at the start of the second sentence of the first paragraph.

To avoid this confusion, explicit `<p>` elements can be used. For example:

```
<section>
  <h2>My Cats</h2>
  <p>You can play with my cat simulator.</p>
  <object data="cats.sim">
    <p>To see the cat simulator, use one of the following links:</p>
    <ul>
      <li><a href="cats.sim">Download simulator file</a>
      <li><a href="https://sims.example.com/watch?v=LYds5xY4INU">Use online simulator</a>
    </ul>
    <p>Alternatively, upgrade to the Mellblom Browser.</p>
  </object>
  <p>I'm quite proud of it.</p>
</section>
```

§ 3.2.5. Global attributes

The following attributes are common to and may be specified on all [html elements](#) (even those not defined in this specification):

- [accesskey](#)
- `class`

- [contenteditable](#)
- [contextmenu](#)
- [dir](#)
- [draggable](#)
- [dropzone](#)
- hidden
- [id](#)
- [lang](#)
- spellcheck
- style
- tabindex
- [title](#)
- translate

These attributes are only defined by this specification as attributes for [HTML elements](#). When this specification refers to elements having these attributes, elements from namespaces that are not defined as having these attributes must not be considered as being elements with these attributes.

EXAMPLE 71

For example, in the following XML fragment, the "bogus" element does not have a [dir](#) attribute as defined in this specification, despite having an attribute with the literal name "[dir](#)". Thus, [the directionality](#) of the inner-most [](#) element is "[rtl](#)", inherited from the [<div>](#) element indirectly through the "bogus" element.

```
<div xmlns="http://www.w3.org/1999/xhtml" dir="rtl">
  <bogus xmlns="https://example.net/ns" dir="ltr">
    <span xmlns="http://www.w3.org/1999/xhtml">
      </span>
    </bogus>
  </div>
```



To enable assistive technology products to expose a more fine-grained interface than is otherwise possible with HTML elements and attributes, a set of annotations for assistive technology products can be specified (the ARIA role and aria-* attributes). [\[WAI-ARIA\]](#)



The following [event handler content attributes](#) may be specified on any [HTML element](#):

- onabort
- onblur*
- oncancel
- oncanplay
- oncanplaythrough
- onchange

- onclick
- onclose
- oncontextmenu
- oncopy
- oncuechange
- oncut
- ondblclick
- ondrag
- ondragend
- ondragenter
- ondragexit
- ondragleave
- ondragover
- ondragstart
- ondrop
- ondurationchange
- onemptied
- onended
- onerror*
- onfocus*
- oninput
- oninvalid
- onkeydown
- onkeypress
- onkeyup
- onload*
- onloadeddata
- onloadedmetadata
- onloadstart
- onmousedown
- onmouseenter
- onmouseleave
- onmousemove
- onmouseout
- onmouseover

- onmouseup
- onwheel
- onpaste
- onpause
- onplay
- onplaying
- onprogress
- onratechange
- onreset
- onresize*
- onscroll*
- onseeked
- onseeking
- onselect
- onshow
- onstalled
- onsubmit
- onsuspend
- ontimeupdate
- ontoggle
- onvolumechange
- onwaiting

NOTE:

The attributes marked with an asterisk have a different meaning when specified on [<body>](#) elements as those elements expose [event handlers](#) of the Window object with the same names.

NOTE:

While these attributes apply to all elements, they are not useful on all elements. For example, only [media elements](#) will ever receive a [volumechange](#) event fired by the user agent.



[Custom data attributes](#) (e.g., `data-foldername` or `data-msgid`) can be specified on any [HTML element](#), to store custom data specific to the page.



In [HTML documents](#), elements in the [HTML namespace](#) may have an `xmlns` attribute specified, if, and only if, it has the exact value `"http://www.w3.org/1999/xhtml"`. This does not apply to [XML documents](#).

NOTE:

In HTML, the `xmlns` attribute has absolutely no effect. It is basically a talisman. It is allowed merely to make migration to and from XHTML mildly easier. When parsed by an [HTML parser](#), the attribute ends up in no namespace, not the `"http://www.w3.org/2000/xmlns/"` namespace like namespace declaration attributes in XML do.

NOTE:

In XML, an `xmlns` attribute is part of the namespace declaration mechanism, and an element cannot actually have an `xmlns` attribute in no namespace specified.



The XML specification also allows the use of the `xml:space` attribute in the [XML namespace](#) on any element in an [XML document](#). This attribute has no effect on [html elements](#), as the default behavior in HTML is to preserve whitespace. [\[XML\]](#)

NOTE:

There is no way to serialize the `xml:space` attribute on [html elements](#) in the [text/html](#) syntax.

§ 3.2.5.1. The `id` attribute

The `id` attribute specifies its element's [unique identifier \(ID\)](#). [\[DOM\]](#)

The value must be unique amongst all the [IDs](#) in the element's [home subtree](#) and must contain at least one character. The value must not contain any [space characters](#).

NOTE:

There are no other restrictions on what form an ID can take; in particular, IDs can consist of just digits, start with a digit, start with an underscore, consist of just punctuation, etc.

NOTE:

An element's [unique identifier](#) can be used for a variety of purposes, most notably as a way to link to specific parts of a document using fragment identifiers, as a way to target an element when scripting, and as a way to style a specific element from CSS.

Identifiers are opaque strings. Particular meanings should not be derived from the value of the `id` attribute.

§ 3.2.5.2. The `title` attribute

The `title` attribute [represents](#) advisory information for the element, such as would be appropriate for a tooltip. On a link, this could be the title or a description of the target resource; on an image, it could be the image credit or a description of the image; on a paragraph, it could be a footnote or commentary on the text; on a citation, it could be further information about the source; on [interactive content](#), it could be a label for, or instructions for, use of the element; and so forth. The value is text.

NOTE:

Relying on the `title` attribute is currently discouraged as many user agents do not expose the attribute in an accessible manner as required by this specification (e.g., requiring a pointing device such as a mouse to cause a tooltip to appear, which excludes keyboard-only users and touch-only users, such as anyone with a modern phone or tablet).

If this attribute is omitted from an element, then it implies that the `title` attribute of the nearest ancestor [HTML element](#) with a `title` attribute set is also relevant to this element. Setting the attribute overrides this, explicitly stating that the advisory information of any ancestors is not relevant to this element. Setting the attribute to the empty string indicates that the element has no advisory information.

If the `title` attribute's value contains U+000A LINE FEED (LF) characters, the content is split into multiple lines. Each U+000A LINE FEED (LF) character represents a line break.

EXAMPLE 72

Caution is advised with respect to the use of newlines in `title` attributes.

For instance, the following snippet actually defines an abbreviation's expansion *with a line break in it*:

```
<p>My logs show that there was some interest in <abbr title="Hypertext
Transport Protocol">HTTP</abbr> today.</p>
```

Some elements, such as `<link>`, `<abbr>`, and `<input>`, define additional semantics for the `title` attribute beyond the semantics described above.

The **advisory information** of an element is the value that the following algorithm returns, with the algorithm being aborted once a value is returned. When the algorithm returns the empty string, then there is no advisory information.

1. If the element is a `<link>`, `<style>`, `<dfn>`, `<abbr>`, or `<menuitem>` element, then: if the element has a `title` attribute, return the value of that attribute, otherwise, return the empty string.
2. Otherwise, if the element has a `title` attribute, then return its value.
3. Otherwise, if the element has a parent element, then return the parent element's [advisory information](#).
4. Otherwise, return the empty string.

User agents should inform the user when elements have [advisory information](#), otherwise the information would not be discoverable.



The `title` IDL attribute must [reflect](#) the `title` content attribute.

§ 3.2.5.3. The `lang` and `xml:lang` attributes

The **lang** attribute (in no namespace) specifies the primary language for the element's contents and for any of the element's attributes that contain text. Its value must be a valid BCP 47 language tag, or the empty string. Setting the attribute to the empty string indicates that the primary language is unknown. [\[BCP47\]](#)

The **lang** attribute in the [XML namespace](#) is defined in XML. [\[XML\]](#)

If these attributes are omitted from an element, then the language of this element is the same as the language of its parent element, if any.

The **lang** attribute in no namespace may be used on any [HTML element](#).

The **lang** attribute in the [XML namespace](#) may be used on [html elements](#) in [XML documents](#), as well as elements in other namespaces if the relevant specifications allow it (in particular, MathML and SVG allow **lang** attributes in the [XML namespace](#) to be specified on their elements). If both the **lang** attribute in no namespace and the **lang** attribute in the [XML namespace](#) are specified on the same element, they must have exactly the same value when compared in an [ASCII case-insensitive](#) manner.

Authors must not use the **lang** attribute in the [XML namespace](#) on [html elements](#) in [HTML documents](#). To ease migration to and from XHTML, authors may specify an attribute in no namespace with no prefix and with the literal localname "`xml:lang`" on [html elements](#) in [HTML documents](#), but such attributes must only be specified if a **lang** attribute in no namespace is also specified, and both attributes must have the same value when compared in an [ASCII case-insensitive](#) manner.

NOTE:

The attribute in no namespace with no prefix and with the literal localname "`xml:lang`" has no effect on language processing.

⚠Warning! The language of [HTML documents](#) is indicated using a **lang** attribute (on the `<HTML>` element itself,

to indicate the primary language of the document, and on individual elements, to indicate a change in language). It provides an explicit indication to user agents about the language of content, so an appropriate language dictionary can be used and, in the case of screen readers and similar assistive technologies with voice output, the content is pronounced using the correct voice / language library (where available). **Setting of a language using the `lang` attribute which does not match the language of the document or document parts will result in some users being unable to understand the content.**



To determine the **language** of a node, user agents must look at the nearest ancestor element (including the element itself if the node is an element) that has a `lang` attribute in the [XML namespace](#) set or is an [HTML element](#) and has a `lang` in no namespace attribute set. That attribute specifies the language of the node (regardless of its value).

If both the `lang` attribute in no namespace and the `lang` attribute in the [XML namespace](#) are set on an element, user agents must use the `lang` attribute in the [XML namespace](#), and the `lang` attribute in no namespace must be [ignored](#) for the purposes of determining the element's language.

If neither the node nor any of the node's ancestors, including the [root element](#), have either attribute set, but there is a [pragma-set default language](#) set, then that is the language of the node. If there is no [pragma-set default language](#) set, then language information from a higher-level protocol (such as HTTP), if any, must be used as the final fallback language instead. In the absence of any such language information, and in cases where the higher-level protocol reports multiple languages, the language of the node is unknown, and the corresponding language tag is the empty string.

EXAMPLE 73

For example, if a document is delivered over HTTP and the Content-Language HTTP header is specified with a value "en" (and there is no pragma-set default language), then for any element in the document that does not itself have a `lang` attribute nor any ancestor of that element, the fallback language for the element will be English. If the value of the Content-Language header was "de, fr, it" then the language of the node is unknown. [This article](#) provides some additional guidance on the use of HTTP headers, and `<meta>` elements for providing language information.

If the resulting value is not a recognized language tag, then it must be treated as an unknown language having the given language tag, distinct from all other languages. For the purposes of round-tripping or communicating with other services that expect language tags, user agents should pass unknown language tags through unmodified, and tagged as being BCP 47 language tags, so that subsequent services do not interpret the data as another type of language description. [\[BCP47\]](#)

EXAMPLE 74

Thus, for instance, an element with `lang="xyzy"` would be matched by the selector `:lang(xyzy)` (e.g., in CSS), but it would not be matched by `:lang(abcde)`, even though both are equally invalid. Similarly, if a Web browser and screen reader working in unison communicated about the language of the element, the browser would tell the screen reader that the language was "xyzy", even if it knew it was invalid, just in case the screen reader actually supported a language with that tag after all. Even if the screen reader supported both BCP 47 and another syntax for encoding language names, and in that other syntax the string "xyzy" was a way to denote the Belarusian language, it would be *incorrect* for the screen reader to then start treating text as Belarusian, because "xyzy" is not how Belarusian is described in BCP 47 codes (BCP 47 uses the code "be" for Belarusian).

If the resulting value is the empty string, then it must be interpreted as meaning that the language of the node is explicitly unknown.



User agents may use the element's language to determine proper processing or rendering (e.g., in the selection of appropriate fonts or pronunciations, for dictionary selection, or for the user interfaces of form controls such as date pickers).



The `lang` IDL attribute must [reflect](#) the `lang` content attribute in no namespace.

§ 3.2.5.4. The `translate` attribute

The **translate** attribute is an [enumerated attribute](#) that is used to specify whether an element's attribute values and the values of its Text node children are to be translated when the page is localized, or whether to leave them unchanged.

The attribute's keywords are the empty string, `yes`, and `no`. The empty string and the `yes` keyword map to the *yes* state. The `no` keyword maps to the *no* state. In addition, there is a third state, the *inherit* state, which is the *missing value default* (and the *invalid value default*).

Each element (even non-HTML elements) has a **translation mode**, which is in either the [translate-enabled](#) state or the [no-translate](#) state. If an [HTML element](#)'s `translate` attribute is in the *yes* state, then the element's [translation mode](#) is in the [translate-enabled](#) state; otherwise, if the element's `translate` attribute is in the *no* state, then the element's [translation mode](#) is in the [no-translate](#) state. Otherwise, either the element's `translate` attribute is in the *inherit* state, or the element is not an [HTML element](#) and thus does not have a `translate` attribute; in either case, the element's [translation mode](#) is in the same state as its parent element's, if any, or in the [translate-enabled](#) state, if the element is a [root element](#).

When an element is in the **translate-enabled** state, the element's [translatable attributes](#) and the values of its Text node children are to be translated when the page is localized.

When an element is in the **no-translate** state, the element's attribute values and the values of its Text node children are to be left as-is when the page is localized, e.g., because the element contains a person's name or a name of a computer program.

The following attributes are **translatable attributes**:

- `abbr` on [<th>](#) elements
- `alt` on [<area>](#), [](#), and [<input>](#) elements
- `content` on [<meta>](#) elements, if the `name` attribute specifies a metadata name whose value is known to be translatable
- `download` on [<a>](#) and [<area>](#) elements
- `label` on [<menuitem>](#), [<menu>](#), [<optgroup>](#), [<option>](#), and [<track>](#) elements
- `lang` on [html elements](#); must be "translated" to match the language used in the translation
- `placeholder` on `input` and [<textarea>](#) elements
- `srcdoc` on [<iframe>](#) elements; must be parsed and recursively processed
- `style` on [html elements](#); must be parsed and recursively processed (e.g., for the values of "content" properties)
- `title` on all [html elements](#)
- `value` on [<input>](#) elements with a `type` attribute in the [Button](#), [submit button](#), or [reset button](#) state



The `translate` IDL attribute must, on getting, return `true` if the element's [translation mode](#) is [translate-enabled](#), and `false` otherwise. On setting, it must set the `content` attribute's value to `"yes"` if the new value is `true`, and set the `content` attribute's value to `"no"` otherwise.

EXAMPLE 75

In this example, everything in the document is to be translated when the page is localized, except the sample keyboard input and sample program output:

```
<!DOCTYPE HTML>
<html> <!-- default on the root element is translate=yes -->
  <head>
    <title>The Bee Game</title> <!-- implied translate=yes inherited from ancestors -->
  </head>
  <body>
    <p>The Bee Game is a text adventure game in English.</p>
    <p>When the game launches, the first thing you should do is type
      <kbd translate=no>eat honey</kbd>. The game will respond with:</p>
    <pre><samp translate=no>Yum yum! That was some good honey!</samp></pre>
  </body>
</html>
```

§ 3.2.5.5. *The xml:base attribute (XML only)*

The `xml:base` attribute is defined in XML Base. [\[XMLBASE\]](#)

The `xml:base` attribute may be used on [html elements](#) of [XML documents](#). Authors must not use the `xml:base` attribute on [html elements](#) in [HTML documents](#).

§ 3.2.5.6. *The dir attribute*

The **dir** attribute specifies the element's text directionality. The attribute is an [enumerated attribute](#) with the following keywords and states:

The ltr keyword, which maps to the ltr state

Indicates that the contents of the element are explicitly directionally isolated left-to-right text.

The rtl keyword, which maps to the rtl state

Indicates that the contents of the element are explicitly directionally isolated right-to-left text.

The auto keyword, which maps to the auto state

Indicates that the contents of the element are explicitly directionally isolated text, but that the direction is to be determined programmatically using the contents of the element (as described below).

NOTE:

The heuristic used by this state is very crude (it just looks at the first character with a strong directionality, in a manner analogous to the Paragraph Level determination in the bidirectional algorithm). Authors are urged to only use this value as a last resort when the direction of the text is truly unknown and no better server-side heuristic can be applied. [\[BIDI\]](#)

NOTE:

For `textarea` and `<pre>` elements, the heuristic is applied on a per-paragraph level.

The attribute has no *invalid value default* and no *missing value default*.



The **directionality** of an element (any element, not just an [HTML element](#)) is either "[ltr](#)" or "[rtl](#)", and is determined as per the first appropriate set of steps from the following list:

- If the element's [dir](#) attribute is in the [ltr](#) state
- If the element is a [root element](#) and the [dir](#) attribute is not in a defined state (i.e., it is not present or has an invalid value)

- ↪ If the element is an `<input>` element whose type attribute is in the [Telephone](#) state, and the `dir` attribute is not in a defined state (i.e., it is not present or has an invalid value)

The [directionality](#) of the element is `"ltr"`.

- ↪ If the element's `dir` attribute is in the `rtl` state

The [directionality](#) of the element is `"rtl"`.

- ↪ If the element is an `<input>` element whose type attribute is in the [Text](#), [Search](#), [Telephone](#), [URL](#), or [E-mail](#) state, and the `dir` attribute is in the `auto` state

- ↪ If the element is a `<textarea>` element and the `dir` attribute is in the `auto` state

If the element's [value](#) contains a character of bidirectional character type AL or R, and there is no character of bidirectional character type L anywhere before it in the element's [value](#), then the [directionality](#) of the element is `"rtl"`. [\[BIDI\]](#)

Otherwise, if the element's [value](#) is not the empty string, or if the element is a [root element](#), the [directionality](#) of the element is `"ltr"`.

Otherwise, the [directionality](#) of the element is the same as the element's parent element's [directionality](#).

- ↪ If the element's `dir` attribute is in the `auto` state

- ↪ If the element is a `<bdi>` element and the `dir` attribute is not in a defined state (i.e., it is not present or has an invalid value)

Find the first character in [tree order](#) that matches the following criteria:

- The character is from a Text node that is a descendant of the element whose [directionality](#) is being determined.
- The character is of bidirectional character type L, AL, or R. [\[BIDI\]](#)
- The character is not in a Text node that has an ancestor element that is a descendant of the element whose [directionality](#) is being determined and that is either:
 - A `<bdi>` element.
 - A `<script>` element.
 - A `<style>` element.
 - A `<textarea>` element.
 - An element with a `dir` attribute in a defined state.

If such a character is found and it is of bidirectional character type AL or R, the [directionality](#) of the element is `"rtl"`.

If such a character is found and it is of bidirectional character type L, the [directionality](#) of the element is `"ltr"`.

Otherwise, if the element is a [root element](#), the [directionality](#) of the element is `"ltr"`.

Otherwise, the [directionality](#) of the element the same as the element's parent element's [directionality](#).

- ↪ If the element has a parent element and the `dir` attribute is not in a defined state (i.e., it is not present or has an invalid value)

The [directionality](#) of the element is the same as the element's parent element's [directionality](#).

NOTE:

Since the `dir` attribute is only defined for [html elements](#), it cannot be present on elements from other namespaces. Thus, elements from other namespaces always just inherit their [directionality](#) from their parent element, or, if they don't have one, default to `"ltr"`.

NOTE:

This attribute [has rendering requirements involving the bidirectional algorithm](#).



The **directionality of an attribute** of an [HTML element](#), which is used when the text of that attribute is to be included in the rendering in some manner, is determined as per the first appropriate set of steps from the following list:

➡ If the attribute is a [directionality-capable attribute](#) and the element's [dir](#) attribute is in the [auto](#) state

Find the first character (in logical order) of the attribute's value that is of bidirectional character type L, AL, or R.

[BIDI](#)

If such a character is found and it is of bidirectional character type AL or R, the [directionality of the attribute](#) is ["rtl"](#).

Otherwise, the [directionality of the attribute](#) is ["ltr"](#).

➡ Otherwise

The [directionality of the attribute](#) is the same as [the element's directionality](#).

The following attributes are **directionality-capable attributes**:

- abbr on [<th>](#) elements
- alt on [<area>](#), [](#), and [<input>](#) elements
- content on [<meta>](#) elements, if the name attribute specifies a metadata name whose value is primarily intended to be human-readable rather than machine-readable
- label on [<menuitem>](#), [<menu>](#), [<optgroup>](#), [<option>](#), and [<track>](#) elements
- placeholder on [<input>](#) and [<textarea>](#) elements
- title on all [html elements](#)



This definition is non-normative. Implementation requirements are given below this definition.

`document.dir [ = value ]`

Returns the [<html>](#) element's [dir](#) attribute's value, if any.

Can be set, to either ["ltr"](#), ["rtl"](#), or ["auto"](#) to replace the [<html>](#) element's [dir](#) attribute's value.

If there is no [<html>](#) element, returns the empty string and ignores new values.

The [dir](#) IDL attribute on an element must [reflect](#) the [dir](#) content attribute of that element, [limited to only known values](#).

The [dir](#) IDL attribute on [Document](#) objects must [reflect](#) the [dir](#) content attribute of the [<html>](#) element, if any, [limited to only known values](#). If there is no such element, then the attribute must return the empty string and do nothing on setting.

NOTE:

Authors are strongly encouraged to use the [dir](#) attribute to indicate text direction rather than using CSS, since that way their documents will continue to render correctly even in the absence of CSS (e.g., as interpreted by search engines).

EXAMPLE 76

This markup fragment is of an IM conversation.

```
<p dir=auto class="u1"><b><bdi>Student</bdi>:</b> How do you write "What's your name?" in Arabic?</p>
<p dir=auto class="u2"><b><bdi>Teacher</bdi>:</b> ما اسمك؟</p>
<p dir=auto class="u1"><b><bdi>Student</bdi>:</b> Thanks.</p>
<p dir=auto class="u2"><b><bdi>Teacher</bdi>:</b> That's written "شكراً".</p>
<p dir=auto class="u2"><b><bdi>Teacher</bdi>:</b> Do you know how to write "Please"?</p>
<p dir=auto class="u1"><b><bdi>Student</bdi>:</b> "من فضلك", right?</p>
```

Given a suitable style sheet and the default alignment styles for the `<p>` element, namely to align the text to the *start edge* of the paragraph, the resulting rendering could be as follows:

Student: How do you write "What's your name?" in Arabic?

Teacher: ما اسمك?

Student: Thanks.

Teacher: That's written "شكراً".

Teacher: Do you know how to write "Please"?

Student: "من فضلك", right?

As noted earlier, the `auto` value is not a panacea. The final paragraph in this example is misinterpreted as being right-to-left text, since it begins with an Arabic character, which causes the "right?" to be to the left of the Arabic text.

3.2.5.7. The `class` attribute

Every [HTML element](#) may have a `class` attribute specified.

The attribute, if specified, must have a value that is a [set of space-separated tokens](#) representing the various classes that the element belongs to.

The classes that an [HTML element](#) has assigned to it consists of all the classes returned when the value of the `class` attribute is [split on spaces](#). (Duplicates are ignored.)

NOTE:

Assigning classes to an element affects class matching in selectors in CSS, the `getElementsByClassName()` method in the DOM, and other such features.

There are no additional restrictions on the tokens authors can use in the `class` attribute, but authors are encouraged to use values that describe the nature of the content, rather than values that describe the desired presentation of the content.

**NOTE:**

The `className` and `classList` IDL attributes, defined in the DOM specification, [reflect](#) the `class` content attribute. [\[DOM\]](#)

3.2.5.8. The `style` attribute

⚠Warning! There are no known native implementations of blocking the `style` content attribute based on CSP3 directives. Therefore this feature should not be relied upon.

All [html elements](#) may have the `style` content attribute set. This is a [CSS styling attribute](#) as defined by the CSS Styling Attribute Syntax specification. [\[CSS-STYLE-ATTR\]](#)

In user agents that support CSS, the attribute's value must be parsed when the attribute is added or has its value changed, according to the rules given for [CSS styling attributes](#). [\[CSS-STYLE-ATTR\]](#)

However, if the [Should element's inline behavior be blocked by Content Security Policy?](#) algorithm returns "Blocked" when executed upon the attribute's element and "style attribute", then the style rules defined in the attribute's value must not be applied to the element. [\[CSP3\]](#)

Documents that use [style](#) attributes on any of their elements must still be comprehensible and usable if those attributes were removed.

NOTE:

In particular, using the [style](#) attribute to hide and show content, or to convey meaning that is otherwise not included in the document, is non-conforming. (To hide and show content, use the `hidden` attribute.)



This definition is non-normative. Implementation requirements are given below this definition.

***element* . style**

Returns a `CSSStyleDeclaration` object for the element's [style](#) attribute.

The `style` IDL attribute is defined in the CSS Object Model (CSSOM) specification. [\[CSSOM\]](#)

EXAMPLE 77

In the following example, the words that refer to colors are marked up using the `<span>` element and the `style` attribute to make those words show up in the relevant colors in visual media.

```
<p>My sweat suit is <span style="color: green; background:
transparent">green</span> and my eyes are <span style="color: blue;
background: transparent">blue</span>.</p>
```

3.2.5.9. Embedding custom non-visible data with the data- attributes*

A **custom data attribute** is an attribute in no namespace whose name starts with the string "**data-**", has at least one character after the hyphen, is [XML-compatible](#), and contains no [uppercase ASCII letters](#).

NOTE:

All attribute names on [html elements](#) in [HTML documents](#) get ASCII-lowercased automatically, so the restriction on ASCII uppercase letters doesn't affect such documents.

[Custom data attributes](#) are intended to store custom data private to the page or application, for which there are no more appropriate attributes or elements.

These attributes are not intended for use by software that is not known to the administrators of the site that uses the attributes. For generic extensions that are to be used by multiple independent tools, either this specification should be extended to provide the feature explicitly, or a technology like microdata should be used (with a standardized vocabulary).

EXAMPLE 78

For instance, a site about music could annotate list items representing tracks in an album with custom data attributes containing the length of each track. This information could then be used by the site itself to allow the user to sort the list by track length, or to filter the list for tracks of certain lengths.

```
<ol>
  <li data-length="2m11s">Beyond The Sea</li>
  ...
</ol>
```

It would be inappropriate, however, for the user to use generic software not associated with that music site to search for tracks of a certain length by looking at this data.

This is because these attributes are intended for use by the site's own scripts, and are not a generic extension mechanism for publicly-usable metadata.

EXAMPLE 79

Similarly, a page author could write markup that provides information for a translation tool that they are intending to use:

```
<p>The third <span data-mytrans-de="Anspruch">claim</span> covers the case of
<span translate="no">HTML</span> markup.</p>
```

In this example, the "data-mytrans-de" attribute gives specific text for the MyTrans product to use when translating the phrase "claim" to German. However, the standard `translate` attribute is used to tell it that in all languages, "HTML" is to remain unchanged. When a standard attribute is available, there is no need for a [custom data attribute](#) to be used.

Every [HTML element](#) may have any number of [custom data attributes](#) specified, with any value.



This definition is non-normative. Implementation requirements are given below this definition.

***element* . dataset**

Returns a `DOMStringMap` object for the element's `data-*` attributes.

Hyphenated names become camel-cased. For example, `data-foo-bar=""` becomes `element.dataset.fooBar`.

The [dataset](#) IDL attribute provides convenient accessors for all the `data-*` attributes on an element. On getting, the `dataset` IDL attribute must return a `DOMStringMap` object, associated with the following algorithms, which expose these attributes on their element:

The algorithm for getting the list of name-value pairs

1. Let *list* be an empty list of name-value pairs.
2. For each content attribute on the element whose first five characters are the string "data-" and whose remaining characters (if any) do not include any [uppercase ASCII letters](#), in the order that those attributes are listed in the element's [attribute list](#), add a name-value pair to *list* whose name is the attribute's name with the first five characters removed and whose value is the attribute's value.
3. For each name in *list*, for each U+002D HYPHEN-MINUS character (-) in the name that is followed by a [lower-case ASCII letter](#), remove the U+002D HYPHEN-MINUS character (-) and replace the character that followed it by the same character [converted to ASCII uppercase](#).
4. Return *list*.

The algorithm for setting names to certain values

1. Let *name* be the name passed to the algorithm.
2. Let *value* be the value passed to the algorithm.
3. If *name* contains a U+002D HYPHEN-MINUS character (-) followed by a [lowercase ASCII letter](#), throw a `"SyntaxError"` [DOMException](#) and abort these steps.
4. For each [uppercase ASCII letter](#) in *name*, insert a U+002D HYPHEN-MINUS character (-) before the character and replace the character with the same character [converted to ASCII lowercase](#).
5. Insert the string `data-` at the front of *name*.
6. Set the value of the attribute with the name *name*, to the value *value*, replacing any previous value if the attribute already existed. If `setAttribute()` would have thrown an exception when setting an attribute with the name *name*, then this must throw the same exception.

The algorithm for deleting names

1. Let *name* be the name passed to the algorithm.
2. For each [uppercase ASCII letter](#) in *name*, insert a U+002D HYPHEN-MINUS character (-) before the character and replace the character with the same character [converted to ASCII lowercase](#).
3. Insert the string `data-` at the front of *name*.
4. Remove the attribute with the name *name*, if such an attribute exists. Do nothing otherwise.

NOTE:

This algorithm will only get invoked by the Web IDL specification for names that are given by the earlier algorithm for getting the list of name-value pairs. [\[WEBIDL\]](#)

EXAMPLE 80

If a Web page wanted an element to represent a space ship, e.g., as part of a game, it would have to use the `class` attribute along with `data-*` attributes:

```
<div class="spaceship" data-ship-id="30">
  <button class="fire" onclick="spaceships[this.parentNode.dataset.shipId].fire()">
    Fire
  </button>
</div>
```

Notice how the hyphenated attribute name becomes camel-cased in the API.

Authors should carefully design such extensions so that when the attributes are ignored and any associated CSS dropped, the page is still usable.

User agents must not derive any implementation behavior from these attributes or values. Specifications intended for user agents must not define these attributes to have any meaningful values.

JavaScript libraries may use the [custom data attributes](#), as they are considered to be part of the page on which they are used. Authors of libraries that are reused by many authors are encouraged to include their name in the attribute names, to reduce the risk of clashes. Where it makes sense, library authors are also encouraged to make the exact name used in the attribute names customizable, so that libraries whose authors unknowingly picked the same name can be used on the same page, and so that multiple versions of a particular library can be used on the same page even when those versions are not mutually compatible.

EXAMPLE 81

For example, a library called "DoQuery" could use attribute names like `data-doquery-range`, and a library called "jJo" could use attributes names like `data-jjo-range`. The jJo library could also provide an API to set which prefix to use (e.g., `J.setDataPrefix("j2")`), making the attributes have names like `data-j2-range`.

§ 3.2.6. Requirements relating to the bidirectional algorithm

§ 3.2.6.1. Authoring conformance criteria for bidirectional-algorithm formatting characters

[Text content](#) in [html elements](#) with Text nodes in their [contents](#), and text in attributes of [html elements](#) that allow free-form text, may contain characters in the ranges U+202A to U+202E and U+2066 to U+2069 (the bidirectional-algorithm formatting characters). However, the use of these characters is restricted so that any embedding or overrides generated by these characters do not start and end with different parent elements, and so that all such embeddings and overrides are explicitly terminated by a U+202C POP DIRECTIONAL FORMATTING character. This helps reduce incidences of text being reused in a manner that has unforeseen effects on the bidirectional algorithm. [\[BIDI\]](#)

The aforementioned restrictions are defined by specifying that certain parts of documents form [bidirectional-algorithm formatting character ranges](#), and then imposing a requirement on such ranges.

The strings resulting from applying the following algorithm to an [HTML element](#) *element* are [bidirectional-algorithm formatting character ranges](#):

1. Let *output* be an empty list of strings.
2. Let *string* be an empty string.
3. Let *node* be the first child node of *element*, if any, or null otherwise.
4. *Loop*: If *node* is null, jump to the step labeled *end*.
5. Process *node* according to the first matching step from the following list:
 - ↪ If *node* is a Text node
 - Append the text data of *node* to *string*.
 - ↪ If *node* is a `<br>` element
 - ↪ If *node* is an [HTML element](#) that is [flow content](#) but that is not also [phrasing content](#)
 - If *string* is not the empty string, push *string* onto *output*, and let *string* be empty string.
 - ↪ Otherwise
 - Do nothing.
6. Let *node* be *node*'s next sibling, if any, or null otherwise.
7. Jump to the step labeled *loop*.
8. *End*: If *string* is not the empty string, push *string* onto *output*.
9. Return *output* as the [bidirectional-algorithm formatting character ranges](#).

The value of a namespace-less attribute of an [HTML element](#) is a [bidirectional-algorithm formatting character range](#).

Any strings that, as described above, are [bidirectional-algorithm formatting character ranges](#) must match the string production in the following ABNF, the character set for which is Unicode. [\[ABNF\]](#)

```
string      = *( plaintext ( embedding / override / isolation ) ) plaintext
embedding  = ( lre / rle ) string pdf
override   = ( lro / rlo ) string pdf
isolation  = ( lri / rli / fsi ) string pdi
lre        = %x202A ; U+202A LEFT-TO-RIGHT EMBEDDING
rle        = %x202B ; U+202B RIGHT-TO-LEFT EMBEDDING
lro        = %x202D ; U+202D LEFT-TO-RIGHT OVERRIDE
rlo        = %x202E ; U+202E RIGHT-TO-LEFT OVERRIDE
pdf        = %x202C ; U+202C POP DIRECTIONAL FORMATTING
lri        = %x2066 ; U+2066 LEFT-TO-RIGHT ISOLATE
rli        = %x2067 ; U+2067 RIGHT-TO-LEFT ISOLATE
fsi        = %x2068 ; U+2068 FIRST STRONG ISOLATE
pdi        = %x2069 ; U+2069 POP DIRECTIONAL ISOLATE
plaintext  = *( %x0000-2029 / %x202F-2065 / %x206A-10FFFF )
              ; any string with no bidirectional-algorithm formatting characters
```

NOTE:

While the U+2069 POP DIRECTIONAL ISOLATE character implicitly also ends open embeddings and overrides, text that relies on this implicit scope closure is not conforming to this specification. All strings of embeddings, overrides, and isolations need to be explicitly terminated to conform to this section's requirements.

NOTE:

Authors are encouraged to use the `dir` attribute, the `<bdo>` element, and the `<bdi>` element, rather than maintaining the bidirectional-algorithm formatting characters manually. The bidirectional-algorithm formatting characters interact poorly with CSS.

§ 3.2.6.2. User agent conformance criteria

User agents must implement the Unicode bidirectional algorithm to determine the proper ordering of characters when rendering documents and parts of documents. [\[BIDI\]](#)

The mapping of HTML to the Unicode bidirectional algorithm must be done in one of three ways. Either the user agent must implement CSS, including in particular the CSS [‘unicode-bidi’](#), [‘direction’](#), and [‘content’](#) properties, and must have, in its user agent style sheet, the rules using those properties given in this specification's [§10 Rendering](#) section, or, alternatively, the user agent must act as if it implemented just the aforementioned properties and had a user agent style sheet that included all the aforementioned rules, but without letting style sheets specified in documents override them, or, alternatively, the user agent must implement another styling language with equivalent semantics. [\[CSS-WRITING-MODES-3\]](#) [\[CSS3-CONTENT\]](#)

The following elements and attributes have requirements defined by the [§10 Rendering](#) section that, due to the requirements in this section, are requirements on all user agents (not just those that [support the suggested default rendering](#)):

- [dir](#) attribute
- [<bdi>](#) element
- [<bdo>](#) element
- [
](#) element
- [<pre>](#) element
- [<textarea>](#) element
- [<wbr>](#) element

§ 3.2.7. WAI-ARIA and HTML Accessibility API Mappings

§ 3.2.7.1. ARIA Authoring Requirements

Authors may use the ARIA `role` and `aria-*` attributes on [HTML elements](#), in accordance with the requirements described in the ARIA specifications, except where these conflict with the requirements specified in ARIA in HTML [\[html-aria\]](#). These exceptions are intended to prevent authors from making assistive technology products report nonsensical states that do not represent the actual state of the document. [\[WAI-ARIA\]](#)

NOTE:

In the majority of cases setting an ARIA `role` and/or `aria-*` attribute that matches the default implicit ARIA semantics is unnecessary and not recommended as these properties are already set by the browser.

Authors are encouraged to make use of the following documents for guidance on using ARIA in HTML beyond that which is provided in this section:

- [Using ARIA](#) - A practical guide for developers on how to add accessibility information to HTML elements using the Accessible Rich Internet Applications specification [\[WAI-ARIA\]](#).
- [WAI-ARIA Authoring Practices 1.1](#) - An author's guide to understanding and implementing Accessible Rich Internet Applications.

§ 3.2.7.2. Conformance Checker Implementation Requirements

Conformance checkers are required to implement document conformance requirements for use of the ARIA role and `aria-*` attributes on [HTML elements](#), as defined in ARIA in HTML. [\[html-aria\]](#)

§ 3.2.7.3. User Agent Implementation Requirements

User agents are required to implement ARIA semantics on all [HTML elements](#), as defined in the ARIA specifications [\[WAI-ARIA\]](#) and [\[core-aam-1.1\]](#).

User agents are required to implement Accessibility API semantics on all [HTML elements](#), as defined in the HTML Accessibility API Mappings specification [\[html-aam-1.0\]](#).

The ARIA attributes defined in the ARIA specifications do not have any effect on CSS pseudo-class matching, user interface modalities that don't use assistive technologies, or the default actions of user interaction events as described in this specification.

§ 3.2.7.3.1. ARIA ROLE ATTRIBUTE

Every HTML element may have an ARIA role attribute specified. This is an ARIA Role attribute as defined by [\[WAI-ARIA\]](#).

The attribute, if specified, must have a value that is a set of space-separated tokens; each token must be a non-abstract role defined in the WAI-ARIA specification [\[WAI-ARIA\]](#).

The WAI-ARIA role that an HTML element has assigned to it is the first non-abstract role found in the list of values generated when the role attribute is split on spaces.

§ 3.2.7.3.2. STATE AND PROPERTY ATTRIBUTES

Every HTML element may have ARIA state and property attributes specified. These attributes are defined by [\[WAI-ARIA\]](#).

A subset of the ARIA State and Property attributes are defined as "**Global States and Properties**" in the [\[WAI-ARIA\]](#) Specification.

These attributes, if specified, must have a value that is the ARIA value type in the "Value" field of the definition for the state or property, mapped to the appropriate HTML value type according to [\[WAI-ARIA\]](#).

ARIA State and Property attributes can be used on any element. They are not always meaningful, however, and in such cases user agents might not perform any processing aside from including them in the DOM. State and property attributes are processed according to the requirements of the HTML Accessibility API Mappings specification [\[html-aam-1.0\]](#), as well as [\[WAI-ARIA\]](#) and , as defined in the ARIA specifications [\[WAI-ARIA\]](#) and [\[core-aam-1.1\]](#).

§ 3.2.7.4. Allowed ARIA roles, states and properties

This section is non-normative.

NOTE:

The following table provides an informative reference to the ARIA roles, states and properties permitted for use in HTML. All ARIA roles, states and properties are normatively defined in the [\[WAI-ARIA\]](#) specification. Links to ARIA roles, states and properties in the table reference the normative [\[WAI-ARIA\]](#) definitions.

ARIA Roles, States and Properties

Role	Description	Required Properties	Supported Properties
any	ARIA global states and properties can be used on any HTML element.	none	• aria-atomic • aria-busy (state) • aria-controls • aria-describedby • aria-disabled (state) • aria-dropeffect • aria-flowto • aria-grabbed (state) • aria-haspopup • aria-hidden (state) • aria-invalid (state) • aria-label • aria-labelledby • aria-live • aria-owns • aria-relevant
alert	A message with important, and usually time-sensitive, information. See related <code>alertdialog</code> and <code>status</code> .	none	• aria-expanded (state)
alertdialog	A type of dialog that contains an alert message, where initial focus goes to an element within the dialog. See related <code>alert</code> and <code>dialog</code> .	none	• aria-expanded (state)
application	A region declared as a web application, as opposed to a web document.	none	• aria-expanded (state)
article	A section of a page that consists of a composition that forms an independent part of a document, page, or site.	none	• aria-expanded (state)
banner	A region that contains mostly site-oriented content, rather than page-specific content.	none	• aria-expanded (state)
button	An input that allows for user-triggered actions when clicked or pressed. See related <code>link</code> .	none	• aria-expanded (state)

Role	Description	Required Properties	Supported Properties
			aria-pressed (state)
checkbox	A checkable input that has three possible values: true, false, or mixed.	aria-checked (state)	
columnheader	A cell containing header information for a column.	none	- aria-sort - aria-readonly - aria-required - aria-selected (state) - aria-expanded (state)
combobox	A presentation of a select; usually similar to a textbox where users can type ahead to select an option, or type to enter arbitrary text as a new item in the list. See related <code>listbox</code> .	aria-expanded (state)	- aria-autocomplete - aria-required - aria-activedescendant
complementary	A supporting section of the document, designed to be complementary to the main content at a similar level in the DOM hierarchy, but remains meaningful when separated from the main content.	none	aria-expanded (state)
contentinfo	A large perceivable region that contains information about the parent document.	none	aria-expanded (state)
definition	A definition of a term or concept.	none	aria-expanded (state)
dialog	A dialog is an application window that is designed to interrupt the current processing of an application in order to prompt the user to enter information or require a response. See related <code>alertdialog</code> .	none	aria-expanded (state)
directory	A list of references to members of a group, such as a static table of contents.	none	aria-expanded (state)
document	A region containing related information that is declared as document content, as opposed to a web application.	none	aria-expanded (state)
form	A landmark region that contains a collection of items and objects that, as a whole, combine to create a form. See related <code>search</code> .	none	aria-expanded (state)
grid	A grid is an interactive control which contains cells of tabular data arranged in rows and columns, like a table.	none	- aria-level - aria-multiselectable

Role	Description	Required Properties	Supported Properties
			- aria-readonly - aria-activedescendant - aria-expanded (state)
gridcell	A cell in a grid or treegrid.	none	- aria-readonly - aria-required - aria-selected (state) - aria-expanded (state)
group	A set of user interface objects which are not intended to be included in a page summary or table of contents by assistive technologies.	none	- aria-activedescendant - aria-expanded (state)
heading	A heading for a section of the page.	none	- aria-level - aria-expanded (state)
img	A container for a collection of elements that form an image.	none	aria-expanded (state)
link	An interactive reference to an internal or external resource that, when activated, causes the user agent to navigate to that resource. See related button.	none	aria-expanded (state)
list	A group of non-interactive list items. See related listbox.	none	aria-expanded (state)
listbox	A widget that allows the user to select one or more items from a list of choices. See related combobox and list.	none	- aria-multiselectable - aria-required - aria-expanded (state) - aria-activedescendant - aria-expanded (state)
listitem	A single item in a list or directory.	none	- aria-level - aria-posinset - aria-setsize - aria-expanded (state)

Role	Description	Required Properties	Supported Properties
log	A type of live region where new information is added in meaningful order and old information may disappear. See related <code>marquee</code> .	none	aria-expanded (state)
main	The main content of a document.	none	aria-expanded (state)
marquee	A type of live region where non-essential information changes frequently. See related <code>log</code> .	none	aria-expanded (state)
math	Content that represents a mathematical expression.	none	aria-expanded (state)
menu	A type of widget that offers a list of choices to the user.	none	- aria-expanded (state) - aria-activedescendant - aria-expanded (state)
menubar	A presentation of menu that usually remains visible and is usually presented horizontally.	none	- aria-expanded (state) - aria-activedescendant - aria-expanded (state)
menuitem	An option in a group of choices contained by a menu or menubar.	none	
menuitemcheckbox	A checkable menuitem that has three possible values: true, false, or mixed.	aria-checked (state)	
menuitemradio	A checkable menuitem in a group of <code>menuitemradio</code> roles, only one of which can be checked at a time.	aria-checked (state)	- aria-posinset - aria-selected (state) - aria-setsize
navigation	A collection of navigational elements (usually links) for navigating the document or related documents.	none	aria-expanded (state)
note	A section whose content is parenthetical or ancillary to the main content of the resource.	none	aria-expanded (state)
option	A selectable item in a select list.	none	- aria-checked (state) - aria-posinset

Role	Description	Required Properties	Supported Properties
			- aria-selected (state) - aria-setsize
presentation	An element whose implicit native role semantics will not be mapped to the accessibility API.	none	
progressbar	An element that displays the progress status for tasks that take a long time.	none	- aria-valuemax - aria-valuemin - aria-valuenow - aria-valuetext
radio	A checkable input in a group of radio roles, only one of which can be checked at a time.	aria-checked (state)	- aria-posinset - aria-selected (state) - aria-setsize
radiogroup	A group of radio buttons.	none	- aria-required - aria-activedescendant - aria-expanded (state)
region	A large perceivable section of a web page or document, that the author feels is important enough to be included in a page summary or table of contents, for example, an area of the page containing live sporting event statistics.	none	aria-expanded (state)
row	A row of cells in a grid.	none	- aria-level - aria-selected (state) - aria-activedescendant - aria-expanded (state)
rowgroup	A group containing one or more row elements in a grid.	none	- aria-activedescendant - aria-expanded (state)
rowheader	A cell containing header information for a row in a grid.	none	- aria-sort - aria-readonly - aria-required - aria-selected (state) - aria-expanded

Role	Description	Required Properties	Supported Properties
			(state)
scrollbar	A graphical object that controls the scrolling of content within a viewing area, regardless of whether the content is fully displayed within the viewing area.	- aria-controls - aria-orientation - aria-valuemax - aria-valuemin - aria-valuenow	aria-expanded (state)
search	A landmark region that contains a collection of items and objects that, as a whole, combine to create a search facility. See related form.	none	- aria-expanded (state) - aria-orientation
separator	A divider that separates and distinguishes sections of content or groups of menuitems.	none	aria-valuetext
slider	A user input where the user selects a value from within a given range.	- aria-valuemax - aria-valuemin - aria-valuenow	- aria-orientation - aria-valuetext
spinbutton	A form of range that expects the user to select from among discrete choices.	- aria-valuemax - aria-valuemin - aria-valuenow	- aria-required - aria-valuetext
status	A container whose content is advisory information for the user but is not important enough to justify an alert, often but not necessarily presented as a status bar. See related alert.	none	aria-expanded (state)
tab	A grouping label providing a mechanism for selecting the tab content that is to be rendered to the user.	none	- aria-selected (state) - aria-expanded (state)
tablist	A list of tab elements, which are references to tabpanel elements.	none	- aria-level - aria-activedescendant - aria-expanded (state)

Role	Description	Required Properties	Supported Properties
tabpanel	A container for the resources associated with a tab, where each tab is contained in a tablist.	none	aria-expanded (state)
textbox	Input that allows free-form text as its value.	none	- aria-activedescendant - aria-autocomplete - aria-multiline - aria-readonly - aria-required
timer	A type of live region containing a numerical counter which indicates an amount of elapsed time from a start point, or the time remaining until an end point.	none	aria-expanded (state)
toolbar	A collection of commonly used function buttons represented in compact visual form.	none	- aria-activedescendant - aria-expanded (state)
tooltip	A contextual popup that displays a description for an element.	none	aria-expanded (state)
tree	A type of list that may contain sub-level nested groups that can be collapsed and expanded.	none	- aria-multiselectable - aria-required - aria-activedescendant - aria-expanded (state)
treegrid	A grid whose rows can be expanded and collapsed in the same manner as for a tree.	none	- aria-level - aria-multiselectable - aria-readonly - aria-activedescendant - aria-expanded (state) - aria-required
treeitem	An option item of a tree. This is an element within a tree that may be expanded or collapsed if it contains a sub-level group of treeitems.	none	- aria-level - aria-posinset - aria-setsize - aria-expanded (state)

Role	Description	Required Properties	Supported Properties
			- aria-checked (state) - aria-selected (state)

§ 4. The elements of HTML

§ 4.1. The root element

§ 4.1.1. The `html` element

Categories:

None.

Contexts in which this element can be used:

As the root element of a document.

Wherever a subdocument fragment is allowed in a compound document.

Content model:

A `<head>` element followed by a `<body>` element.

Tag omission in text/html:

An `<html>` element's [start tag](#) can be omitted if the first thing inside the `<html>` element is not a [comment](#).

An `<html>` element's [end tag](#) can be omitted if the `<html>` element is not immediately followed by a [comment](#).

Content attributes:

[Global attributes](#)

Allowed [ARIA role attribute](#) values:

None

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

DOM interface:

```
interface HTMLHtmlElement : HTMLElement {};
```

The `<html>` element [represents](#) the root of an HTML document.

Authors are encouraged to specify a [lang](#) attribute on the root `<html>` element, giving the document's language. This aids speech synthesis tools to determine what pronunciations to use, translation tools to determine what rules to use, and so forth.

NOTE:

It is recommended to keep the usage of attributes and their values defined on the `<html>` element to a minimum to allow for proper detection of the [character encoding declaration](#) within the first 1024 bytes.

EXAMPLE 82

The `<html>` element in the following example declares that the document's language is English.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Swapping Songs</title>
  </head>
  <body>
    <h1>Swapping Songs</h1>
    <p>Tonight I swapped some of the songs I wrote with some friends, who
      gave me some of the songs they wrote. I love sharing my music.</p>
  </body>
</html>
```

4.2. Document metadata

4.2.1. The head element

Categories:

None.

Contexts in which this element can be used:

As the first element in an `<html>` element.

Content model:

If the document is an [iframe srcdoc document](#) or if title information is available from a higher-level protocol:

Zero or more elements of [metadata content](#), of which no more than one is a `<title>` element and no more than one is a `<base>` element.

Otherwise: One or more elements of [metadata content](#), of which exactly one is a `<title>` element and no more than one is a `<base>` element.

Tag omission in text/html:

A `<head>` element's [start tag](#) may be omitted if the element is empty, or if the first thing inside the `<head>` element is an element.

A `<head>` element's [end tag](#) may be omitted if the `<head>` element is not immediately followed by a [space character](#) or a [comment](#).

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

None

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

DOM interface:

```
interface HTMLHeadElement : HTMLElement {};
```

The `<head>` element [represents](#) a collection of metadata for the [Document](#).

EXAMPLE 83

The collection of metadata in a [<head>](#) element can be large or small. Here is an example of a very short one:

```
<!doctype html>
<html>
  <head>
    <title>A document with a short head</title>
  </head>
  <body>
    ...
```

Here is an example of a longer one:

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
  <META CHARSET="UTF-8">
  <BASE HREF="https://www.example.com/">
  <TITLE>An application with a long head</TITLE>
  <LINK REL="STYLESHEET" HREF="default.css">
  <LINK REL="STYLESHEET ALTERNATE" HREF="big.css" TITLE="Big Text">
  <SCRIPT SRC="support.js"></SCRIPT>
  <META NAME="APPLICATION-NAME" CONTENT="Long headed application">
</HEAD>
<BODY>
  ...
```

NOTE:

The [<title>](#) element is a required child in most situations, but when a higher-level protocol provides title information, e.g., in the Subject line of an e-mail when HTML is used as an e-mail authoring format, the [<title>](#) element can be omitted.

NOTE:

It is recommended to keep the usage of attributes and their values defined on the [<head>](#) element to a minimum to allow for proper detection of the [character encoding declaration](#) within the first 1024 bytes.

4.2.2. The title element**Categories:**

[Metadata content.](#)

Contexts in which this element can be used:

In a [<head>](#) element containing no other [<title>](#) elements.

Content model:

[Text](#) that is not [inter-element whitespace](#).

Tag omission in text/html:

Neither tag is omissible.

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

None

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

DOM interface:

```
interface HTMLTitleElement : HTMLElement {
    attribute DOMString text;
};
```

The `<title>` element [represents](#) the document’s title or name. Authors should use titles that identify their documents even when they are used out of context, for example in a user’s history or bookmarks, or in search results. The document’s title is often different from its first heading, since the first heading does not have to stand alone when taken out of context.

There must be no more than one `<title>` element per document.

NOTE:

If it’s reasonable for the [Document](#) to have no title, then the `title` element is probably not required. See the `<head>` element’s content model for a description of when the element is required.

This definition is non-normative. Implementation requirements are given below this definition.

`title` . `text` [= `value`]

Returns the contents of the element, ignoring child nodes that aren’t Text nodes.

Can be set, to replace the element’s children with the given value.

The IDL attribute `text` must return a concatenation of the contents of all the Text nodes that are children of the `<title>` element (ignoring any other nodes such as comments or elements), in [tree order](#). On setting, it must act the same way as the `textContent` IDL attribute.

EXAMPLE 84

Here are some examples of appropriate titles, contrasted with the top-level headings that might be used on those same pages.

```
<title>Introduction to The Mating Rituals of Bees</title>
...
<h1>Introduction</h1>
<p>This companion guide to the highly successful
<cite>Introduction to Medieval Bee-Keeping</cite> book is...
```

The next page might be a part of the same site. Note how the title describes the subject matter unambiguously, while the first heading assumes the reader knows what the context is and therefore won’t wonder if the dances are Salsa or Waltz:

```
<title>Dances used during bee mating rituals</title>
...
<h2>The Dances</h2>
```

The string to use as the document’s title is given by the `document.title` IDL attribute.

User agents should use the document’s title when referring to the document in their user interface. When the contents of a `<title>` element are used in this way, [the directionality](#) of that `<title>` element should be used to set the directionality of the document’s title in the user interface.

§ 4.2.3. The base element

Categories:

[Metadata content](#).

Contexts in which this element can be used:

In a `<head>` element containing no other `<base>` elements.

Content model:

[Nothing](#).

Tag omission in text/html:

No end tag.

Content attributes:

Global attributes

href — [Document base URL](#)

target — Default [browsing context](#) for [hyperlink navigation](#) and [§4.10.22 Form submission](#)

Allowed [ARIA role attribute](#) values:

None

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#).

DOM interface:

```
interface HTMLBaseElement : HTMLElement {
    attribute DOMString href;
    attribute DOMString target;
};
```

The `<base>` element allows authors to specify the [document base URL](#) for the purposes of [parsing relative URLs](#), and the name of the default [browsing context](#) for the purposes of [following hyperlinks](#). The element does not [represent](#) any content beyond this information.

There must be no more than one `<base>` element per document.

A `<base>` element must have either an [href](#) attribute, a [target](#) attribute, or both.

The [href](#) content attribute, if specified, must contain a [valid URL potentially surrounded by spaces](#).

A `<base>` element, if it has an [href](#) attribute, must come before any other elements in the tree that have attributes defined as taking [URLs](#) except the `<html>` element.

NOTE:

If there are multiple `<base>` elements with [href](#) attributes, all but the first are ignored.

The [target](#) attribute, if specified, must contain a [valid browsing context name or keyword](#), which specifies which [browsing context](#) is to be used as the default when [hyperlinks](#) and [forms](#) in the [Document](#) cause [navigation](#).

A `<base>` element, if it has a [target](#) attribute, must come before any elements in the tree that represent [hyperlinks](#).

NOTE:

If there are multiple `<base>` elements with [target](#) attributes, all but the first are ignored.

A `<base>` element that is the first `<base>` element with an [href](#) content attribute in a particular [Document](#) has a **frozen base URL**. The [frozen base URL](#) must be [immediately set](#) for an element whenever any of the following situations occur:

- The `<base>` element becomes the first `<base>` element in [tree order](#) with an [href](#) content attribute in its [Document](#).
- The `<base>` element is the first `<base>` element in [tree order](#) with an [href](#) content attribute in its [Document](#), and its [href](#) content attribute is changed.

To **set the frozen base URL**, for an element *element*:

1. Let *document* be *element*'s [node document](#).
2. Let *urlRecord* be the result of [parsing](#) the value of *element*'s [href](#) content attribute with *document*'s [fallback base URL](#), and *document*'s [character encoding](#). (Thus the `<base>` element isn't affected by itself.)
3. Set *elements*' [frozen base URL](#) to *document*'s [fallback base URL](#), if *urlRecord* is failure or running [Is base allowed for Document?](#) on the [resulting URL record](#) and *document* returns "Blocked", and to *urlRecord* otherwise.

The [href](#) IDL attribute, on getting, must return the result of running the following algorithm:

1. Let *document* be *element*'s [node document](#).
2. Let *url* be the value of the [href](#) attribute of the `<base>` element, if it has one, and the empty string otherwise.
3. Let *urlRecord* be the result of [parsing url](#) with *document*'s [fallback base url](#), and *document*'s [character encoding](#). (Thus, the `<base>` element isn't affected by other `<base>` elements or itself).
4. If *urlRecord* is failure, return *url*.
5. Return the [serialization](#) of *urlRecord*.

The [href](#) IDL attribute, on setting, must set the [href](#) content attribute to the given new value.

The [target](#) IDL attribute must [reflect](#) the content attribute of the same name.

EXAMPLE 85

In this example, a `<base>` element is used to set the [document base URL](#):

```
<!DOCTYPE html>
<html>
  <head>
    <title>This is an example for the &lt;base&gt; element</title>
    <base href="https://www.example.com/news/index.html">
  </head>
  <body>
    <p>Visit the <a href="archives.html">archives</a>.</p>
  </body>
</html>
```

The link in the above example would be a link to "https://www.example.com/news/archives.html".

4.2.4. The link element

Categories:

[Metadata content](#).

Contexts in which this element can be used:

Where [metadata content](#) is expected.

In a `<noscript>` element that is a child of a `<head>` element.

Content model:

[Nothing](#).

Tag omission in text/html:

No [end tag](#).

Content attributes:

[Global attributes](#)

href — Address of the [hyperlink](#)

crossorigin — How the element handles crossorigin requests

rel — Relationship of this document (or subsection/topic) to the destination resource

rev — [Reverse link](#) relationship of the destination resource to this document (or subsection/topic)

media — Applicable media

hreflang — Language of the linked resource

type — Hint for the type of the referenced resource

sizes — Sizes of the icons (for **rel**="icon")

Also, the [title](#) attribute has special semantics on this element: Title of the link; alternative style sheet set name.

Allowed ARIA role attribute values:

[link](#) (default - *do not set*).

Allowed ARIA state and property attributes:

Global aria-* attributes

Any aria-* attributes [applicable to the allowed roles](#).

For role value

DOM interface:

```
interface HTMLLinkElement : HTMLElement {
  attribute DOMString href;
  attribute DOMString? crossOrigin;
  attribute DOMString rel;
  attribute DOMString rev;
  [SameObject, PutForwards=value] readonly attribute DOMTokenList relList;
  attribute DOMString media;
  attribute DOMString hreflang;
  attribute DOMString type;
  [SameObject, PutForwards=value] readonly attribute DOMTokenList sizes;
};
HTMLLinkElement implements LinkStyle;
```

The `<link>` element allows authors to link their document to other resources.

The destination of the link(s) is given by the **href** attribute, which must be present and must contain a [valid non-empty URL potentially surrounded by spaces](#). If the **href** attribute is absent, then the element does not define a link.

A `<link>` element must have a **rel** attribute.

NOTE:

If the **rel** attribute is used, the element is restricted to the head element.

The types of link indicated (the relationships) are given by the value of the **rel** attribute, which, if present, must have a value that is a [set of space-separated tokens](#). The [allowed keywords and their meanings](#) are defined in a later section. If the **rel** attribute is absent, has no keywords, or if none of the keywords used are allowed according to the definitions in this specification, then the element does not create any links.

Two categories of links can be created using the `<link>` element: [Links to external resources](#) and [hyperlinks](#). The [§4.8.6 Link types](#) section defines whether a particular link type is an external resource or a hyperlink. One link element can create multiple links (of which some might be external resource links and some might be hyperlinks); exactly which and how many links are created depends on the keywords given in the **rel** attribute. User agents must process the links on a per-link basis, not a per-element basis.

NOTE:

Each link created for a `<link>` element is handled separately. For instance, if there are two `<link>` elements with **rel**="stylesheet", they each count as a separate external resource, and each is affected by its own attributes independently. Similarly, if a single `<link>` element has a **rel** attribute with the value `next stylesheet`, it creates both a [hyperlink](#) (for the `next` keyword) and an [external resource link](#) (for the `stylesheet` keyword), and they are affected by other attributes (such as **media** or **title**) differently.

EXAMPLE 86

For example, the following `<link>` element creates two hyperlinks (to the same page):

```
<link rel="author license" href="/about">
```

The two links created by this element are one whose semantic is that the target page has information about the current page's author, and one whose semantic is that the target page has information regarding the license under which the current page is provided.

`<link>` and `<a>` elements may also have a **rev** attribute, which is used to describe a [reverse link](#) relationship from the resource specified by the **href** to the current document. If present, the value of this attribute must be a [set of space-separated](#)

[tokens](#). Like the [rel](#) attribute, §4.8.6 [Link types](#) describes the [allowed keywords and their meanings](#) for the [rev](#) attribute. Both the [rel](#) and [rev](#) attributes may be present on the same element.

Reverse links are a way to express the reverse directional relationship of a link. In contrast to the [rel](#) attribute, whose value conveys a forward directional relationship ("how is the link related to me"), the [rev](#) attribute allows for similar relationships to be expressed in the reverse direction ("how am I related to this link"). These values can enable user agents to build a more comprehensive map of linked documents.

EXAMPLE 87

Given two documents, each containing a chapter of a book, the links between them could be described with the [rel](#) and [rev](#) attributes as follows:

Document with URL "chapter1.html"

```
<link href="chapter2.html" rel="next" rev="prev">
```

Document with URL "chapter2.html"

```
<link href="chapter1.html" rel="prev" rev="next">
<link href="chapter3.html" rel="next" rev="prev">
```

From chapter1.html, the link to chapter2.html is the "next" chapter in the series in the forward direction, and the "previous" chapter in the reverse direction (from chapter2.html to chapter1.html).

EXAMPLE 88

The links in a table of contents document might be described using [rel](#) and [rev](#) as follows:

```
<ol>
  <li><a href="chapter1.html" rev="toc" rel="next">chapter 1</a></li>
  <li><a href="chapter2.html" rev="toc"></a>chapter 2</li>
  <li><a href="chapter3.html" rev="toc"></a>chapter 3</li>
</ol>
```

From the table of contents, the "next" logical path is to the first chapter, expressed using [rel](#). Each chapter link has a "toc" [rev](#) value which indicates that the current document is the table of contents document for every chapter.

The **crossorigin** attribute is a [CORS settings attribute](#). It is intended for use with external resource links.

The exact behavior for links to external resources depends on the exact relationship, as defined for the relevant link type. Some of the attributes control whether or not the external resource is to be applied (as defined below).

For external resources that are represented in the DOM (for example, style sheets), the DOM representation must be made available (modulo cross-origin restrictions) even if the resource is not applied. To **obtain the resource**, the user agent must run the following steps:

1. If the [href](#) attribute's value is the empty string, then abort these steps.
2. [Parse](#) the [URL](#) given by the [href](#) attribute, relative to the element's [node document](#). If that fails, then abort these steps. Otherwise, let *url* be the [resulting URL record](#).
3. Let *corsAttributeState* be the current state of the element's [crossorigin](#) content attribute.
4. Let *request* be the result of [creating a potential-CORS request](#) given *url* and *corsAttributeState*.
5. Set *request*'s [client](#) to the [<link>](#) element's [node document](#)'s [Window](#) object's [environment settings object](#).
6. [Fetch](#) *request*.

User agents may opt to only try to obtain such resources when they are needed, instead of pro-actively fetching all the exter-

nal resources that are not applied.

The semantics of the protocol used (e.g., HTTP) must be followed when fetching external resources. (For example, redirects will be followed and 404 responses will cause the external resource to not be applied.)

Once the attempts to obtain the resource and its [critical subresources](#) are complete, the user agent must, if the loads were successful, [queue a task to fire a simple event](#) named `load` at the `<link>` element, or, if the resource or one of its [critical subresources](#) failed to completely load for any reason (e.g., DNS error, HTTP 404 response, a connection being prematurely closed, unsupported Content-Type), [queue a task to fire a simple event](#) named `error` at the `<link>` element. Non-network errors in processing the resource or its subresources (e.g., CSS parse errors, PNG decoding errors) are not failures for the purposes of this paragraph.

The [task source](#) for these [tasks](#) is the [DOM manipulation task source](#).

The element must [delay the load event](#) of the element's [node document](#) until all the attempts to obtain the resource and its [critical subresources](#) are complete. (Resources that the user agent has not yet attempted to obtain, e.g., because it is waiting for the resource to be needed, do not [delay the load event](#).)



Interactive user agents may provide users with a means to [follow the hyperlinks](#) created using the `<link>` element, somewhere within their user interface. The exact interface is not defined by this specification, but it could include the following information (obtained from the element's attributes, again as defined below), in some form or another (possibly simplified), for each hyperlink created with each `<link>` element in the document:

- The relationship between this document and the resource (given by the [rel](#) attribute)
- The title of the resource (given by the `title` attribute).
- The address of the resource (given by the [href](#) attribute).
- The language of the resource (given by the `hreflang` attribute).
- The optimum media for the resource (given by the `media` attribute).

User agents could also include other information, such as the type of the resource (as given by the `type` attribute).

NOTE:

Hyperlinks created with the `<link>` element and its [rel](#) attribute apply to the whole page. This contrasts with the [rel](#) attribute of `<a>` and `<area>` elements, which indicates the type of a link whose context is given by the link's location within the document.

The `media` attribute says which media the resource applies to. The value must be a [valid media query list](#).

If the link is a [hyperlink](#) then the `media` attribute is purely advisory, and describes for which media the document in question was designed.

However, if the link is an [external resource link](#), then the `media` attribute is prescriptive. The user agent must apply the external resource when the `media` attribute's value [matches the environment](#) and the other relevant conditions apply, and must not apply it otherwise.

NOTE:

The external resource might have further restrictions defined within that limit its applicability. For example, a CSS style sheet might have some `@media` blocks. This specification does not override such further restrictions or requirements.

The default, if the `media` attribute is omitted, is "all", meaning that by default links apply to all media.

The `hreflang` attribute on the `<link>` element has the same semantics as the `hreflang` attribute on the `<a>` element.

The `type` attribute gives the [MIME type](#) of the linked resource. It is purely advisory. The value must be a [valid mime type](#).

For [external resource links](#), the `type` attribute is used as a hint to user agents so that they can avoid fetching resources they

do not support. If the attribute is present, then the user agent must assume that the resource is of the given type (even if that is not a [valid mime type](#), e.g., the empty string). If the attribute is omitted, but the external resource link type has a default type defined, then the user agent must assume that the resource is of that type. If the user agent does not support the given [MIME type](#) for the given link relationship, then the user agent should not [obtain](#) the resource; if the user agent does support the given [MIME type](#) for the given link relationship, then the user agent should [obtain](#) the resource at the appropriate time as specified for the [external resource link](#)'s particular type. If the attribute is omitted, and the external resource link type does not have a default type defined, but the user agent would [obtain](#) the resource if the type was known and supported, then the user agent should [obtain](#) the resource under the assumption that it will be supported.

User agents must not consider the `type` attribute authoritative — upon fetching the resource, user agents must not use the `type` attribute to determine its actual type. Only the actual type (as defined in the next paragraph) is used to determine whether to [apply](#) the resource, not the aforementioned assumed type.

If the external resource link type defines rules for processing the resource's [Content-Type metadata](#), then those rules apply. Otherwise, if the resource is expected to be an image, user agents may apply the [image sniffing rules](#), with the *official type* being the type determined from the resource's [Content-Type metadata](#), and use the resulting [computed type of the resource](#) as if it was the actual type. Otherwise, if neither of these conditions apply or if the user agent opts not to apply the image sniffing rules, then the user agent must use the resource's [Content-Type metadata](#) to determine the type of the resource. If there is no type metadata, but the external resource link type has a default type defined, then the user agent must assume that the resource is of that type.

NOTE:

The stylesheet link type defines rules for processing the resource's [Content-Type metadata](#).

Once the user agent has established the type of the resource, the user agent must apply the resource if it is of a supported type and the other relevant conditions apply, and must ignore the resource otherwise.

EXAMPLE 89

If a document contains style sheet links labeled as follows:

```
<link rel="stylesheet" href="A" type="text/plain">
<link rel="stylesheet" href="B" type="text/css">
<link rel="stylesheet" href="C">
```

...then a compliant user agent that supported only CSS style sheets would fetch the B and C files, and skip the A file (since `text/plain` is not the [MIME type](#) for CSS style sheets).

For files B and C, it would then check the actual types returned by the server. For those that are sent as `text/css`, it would apply the styles, but for those labeled as `text/plain`, or any other type, it would not.

If one of the two files was returned without a [Content-Type](#) metadata, or with a syntactically incorrect type like `Content-Type: "null"`, then the default type for `stylesheet` links would kick in. Since that default type is `text/css`, the style sheet *would* nonetheless be applied.

The **title** attribute gives the title of the link. With one exception, it is purely advisory. The value is text. The exception is for style sheet links, where the **title** attribute defines [alternative style sheet sets](#).

NOTE:

The **title** attribute on `<link>` elements differs from the global **title** attribute of most other elements in that a link without a title does not inherit the title of the parent element: it merely has no title.

The **sizes** attribute is used with the `icon` link type. The attribute must not be specified on `<link>` elements that do not have a **rel** attribute that specifies the `icon` keyword.

The [activation behavior](#) of link elements that create [hyperlinks](#) is to run the following steps:

1. If the `<link>` element's [node document](#) is not [fully active](#), then abort these steps.
2. [Follow the hyperlink](#) created by the `link` element.

HTTP Link: headers, if supported, must be assumed to come before any links in the document, in the order that they were given in the HTTP message. These headers are to be processed according to the rules given in the relevant specifications.

[\[HTTP\]](#) [\[RFC5988\]](#)

NOTE:

Registration of relation types in HTTP Link: headers is distinct from [HTML link types](#), and thus their semantics can be different from same-named HTML types.

The IDL attributes **href**, **rel**, **rev**, **media**, **hreflang**, **type**, and **sizes** each must [reflect](#) the respective content attributes of the same name.

The **crossOrigin** IDL attribute must [reflect](#) the crossorigin content attribute.

The IDL attribute **relList** must [reflect](#) the **rel** content attribute.

relList's DOMTokenList's [supported tokens](#) are the keywords defined in [HTML link types](#) which are allowed on link elements and supported by the user agent.

rel's [supported tokens](#) are the keywords defined in [HTML link types](#) which are allowed on link elements, impact the processing model, and are supported by the user agent. The possible supported tokens are [alternate](#), [dns-prefetch](#), [icon](#), [pre-connect](#), [prefetch](#), [prerender](#), and [stylesheet](#). **rel**'s [supported tokens](#) must only include the tokens from this list that the user agent implements the processing model for.

Other specifications may add [HTML link types](#) as defined in [Other link types](#), such as [\[RESOURCE-HINTS\]](#). These specifications may require that their link types be included in **rel**'s supported tokens.

The LinkStyle interface is also implemented by this element. [\[CSSOM\]](#)

EXAMPLE 90

Here, a set of [<link>](#) elements provide some style sheets:

```
<!-- a persistent style sheet -->
<link rel="stylesheet" href="default.css">

<!-- the preferred alternate style sheet -->
<link rel="stylesheet" href="green.css" title="Green styles">

<!-- some alternate style sheets -->
<link rel="alternate stylesheet" href="contrast.css" title="High contrast">
<link rel="alternate stylesheet" href="big.css" title="Big fonts">
<link rel="alternate stylesheet" href="wide.css" title="Wide screen">
```

EXAMPLE 91

The following example shows how you can specify versions of the page that use alternative formats, are aimed at other languages, and that are intended for other media:

```
<link rel=alternate href="/en/html" hreflang=en type=text/html title="English HTML">
<link rel=alternate href="/fr/html" hreflang=fr type=text/html title="French HTML">
<link rel=alternate href="/en/html/print" hreflang=en type=text/html media=print
title="English HTML (for printing)">
<link rel=alternate href="/fr/html/print" hreflang=fr type=text/html media=print title="French
HTML (for printing)">
<link rel=alternate href="/en/pdf" hreflang=en type=application/pdf title="English PDF">
<link rel=alternate href="/fr/pdf" hreflang=fr type=application/pdf title="French PDF">
```

§ 4.2.5. The meta element

Categories:

[Metadata content.](#)

Contexts in which this element can be used:

If the `charset` attribute is present, or if the element's `http-equiv` attribute is in the [encoding declaration state](#): in a `<head>` element.

If the `http-equiv` attribute is present but not in the [encoding declaration state](#): in a `<head>` element.

If the `http-equiv` attribute is present but not in the [encoding declaration state](#): in a `<noscript>` element that is a child of a `<head>` element.

If the `name` attribute is present: where [metadata content](#) is expected.

Content model:

[Nothing](#).

Tag omission in text/html:

No [end tag](#).

Content attributes:

[Global attributes](#)

`name` — Metadata name

`http-equiv` — Pragma directive

`content` — Value of the element

`charset` — [Character encoding declaration](#)

Allowed [ARIA role attribute](#) values:

None

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

DOM interface:

```
interface HTMLMetaElement : HTMLElement {
    attribute DOMString name;
    attribute DOMString httpEquiv;
    attribute DOMString content;
};
```

The `<meta>` element [represents](#) various kinds of metadata that cannot be expressed using the `<title>`, `<base>`, `<link>`, `<style>`, and `<script>` elements.

The `<meta>` element can represent document-level metadata with the `name` attribute, pragma directives with the **`http-equiv`** attribute, and the file's [character encoding declaration](#) when an HTML document is serialized to string form (e.g., for transmission over the network or for disk storage) with the `charset` attribute.

Exactly one of the `name`, `http-equiv`, and `charset` attributes must be specified.

If either `name` or `http-equiv` is specified, then the `content` attribute must also be specified. Otherwise, it must be omitted.

The **`charset`** attribute specifies the character encoding used by the document. This is a [character encoding declaration](#). If the attribute is present in an [XML document](#), its value must be an [ASCII case-insensitive](#) match for the string "utf-8" (and the document is therefore forced to use [UTF-8](#) as its encoding).

NOTE:

The `charset` attribute on the `<meta>` element has no effect in XML documents, and is only allowed in order to facilitate migration to and from XHTML.

There must not be more than one `<meta>` element with a `charset` attribute per document.

The **`content`** attribute gives the value of the document metadata or pragma directive when the element is used for those purposes. The allowed values depend on the exact context, as described in subsequent sections of this specification.

If a `<meta>` element has a **`name`** attribute, it sets document metadata. Document metadata is expressed in terms of name-value pairs, the `name` attribute on the `<meta>` element giving the name, and the `content` attribute on the same element giving the value. The name specifies what aspect of metadata is being set; valid names and the meaning of their values are described in

the following sections. If a `<meta>` element has no content attribute, then the value part of the metadata name-value pair is the empty string.

The **name** and **content** IDL attributes must [reflect](#) the respective content attributes of the same name. The IDL attribute **httpEquiv** must [reflect](#) the content attribute [http-equiv](#).

§ 4.2.5.1. Standard metadata names

This specification defines a few names for the name attribute of the `<meta>` element.

Names are case-insensitive, and must be compared in an [ASCII case-insensitive](#) manner.

application-name

The value must be a short free-form string giving the name of the Web application that the page represents. If the page is not a Web application, the application-name metadata name must not be used. Translations of the Web application's name may be given, using the [lang](#) attribute to specify the language of each name.

There must not be more than one `<meta>` element with a given [language](#) and with its name attribute set to the value application-name per document.

User agents may use the application name in UI in preference to the page's `<title>`, since the title might include status messages and the like relevant to the status of the page at a particular moment in time instead of just being the name of the application.

To find the application name to use given an ordered list of languages (e.g., British English, American English, and English), user agents must run the following steps:

1. Let *languages* be the list of languages.
2. Let *default language* be the [language](#) of the [Document](#)'s [root element](#), if any, and if that language is not unknown.
3. If there is a *default language*, and if it is not the same language as any of the languages in *languages*, append it to *languages*.
4. Let *winning language* be the first language in *languages* for which there is a `<meta>` element in the [Document](#) that has its name attribute set to the value application-name and whose [language](#) is the language in question.

If none of the languages have such a `<meta>` element, then abort these steps; there's no given application name.

5. Return the value of the content attribute of the first meta element in the [Document](#) in [tree order](#) that has its name attribute set to the value application-name and whose [language](#) is *winning language*.

NOTE:

This algorithm would be used by a browser when it needs a name for the page, for instance, to label a bookmark. The languages it would provide to the algorithm would be the user's preferred languages.

author

The value must be a free-form string giving the name of one of the page's authors.

description

The value must be a free-form string that describes the page. The value must be appropriate for use in a directory of pages, e.g., in a search engine. There must not be more than one `<meta>` element with its name attribute set to the value description per document.

generator

The value must be a free-form string that identifies one of the software packages used to generate the document. This value must not be used on pages whose markup is not generated by software, e.g., pages whose markup was written by a user in a text editor.

EXAMPLE 92

Here is what a tool called "Frontweaver" could include in its output, in the page's `<head>` element, to identify itself as the tool used to generate the page:

```
<meta name=generator content="Frontweaver 8.2">
```

keywords

The value must be a [set of comma-separated tokens](#), each of which is a keyword relevant to the page.

EXAMPLE 93

This page about typefaces on British motorways uses a `<meta>` element to specify some keywords that users might use to look for the page:

```
<!DOCTYPE HTML>
<html lang="en-GB">
  <head>
    <title>Typefaces on UK motorways</title>
    <meta name="keywords" content="british,type face,font,fonts,highway,highways">
  </head>
  <body>
    ...
```

NOTE:

Many search engines do not consider such keywords, because this feature has historically been used unreliably and even misleadingly as a way to spam search engine results in a way that is not helpful for users.

To obtain the list of keywords that the author has specified as applicable to the page, the user agent must run the following steps:

1. Let *keywords* be an empty list.
2. For each `<meta>` element with a name attribute and a content attribute and whose name attribute's value is *keywords*, run the following substeps:
 1. [Split the value of the element's content attribute on commas](#).
 2. Add the resulting tokens, if any, to *keywords*.
3. Remove any duplicates from *keywords*.
4. Return *keywords*. This is the list of keywords that the author has specified as applicable to the page.

User agents should not use this information when there is insufficient confidence in the reliability of the value.

EXAMPLE 94

For instance, it would be reasonable for a content management system to use the keyword information of pages within the system to populate the index of a site-specific search engine, but a large-scale content aggregator that used this information would likely find that certain users would try to game its ranking mechanism through the use of inappropriate keywords.

4.2.5.2. Other metadata names

Extensions to the predefined set of metadata names may be registered in the [WHATWG Wiki MetaExtensions page](#). [\[WHATGWWIKI\]](#)

Anyone is free to edit the WHATWG Wiki MetaExtensions page at any time to add a type. These new names must be specified with the following information:

Keyword

The actual name being defined. The name should not be confusingly similar to any other defined name (e.g., differing only in case).

Brief description

A short non-normative description of what the metadata name's meaning is, including the format the value is required to be in.

Specification

A link to a more detailed description of the metadata name's semantics and requirements. It could be another page on the Wiki, or a link to an external page.

Synonyms

A list of other names that have exactly the same processing requirements. Authors should not use the names defined to be synonyms, they are only intended to allow user agents to support legacy content. Anyone may remove synonyms that are not used in practice; only names that need to be processed as synonyms for compatibility with legacy content are to be registered in this way.

Status

One of the following:

Proposed

The name has not received wide peer review and approval. Someone has proposed it and is, or soon will be, using it.

Ratified

The name has received wide peer review and approval. It has a specification that unambiguously defines how to handle pages that use the name, including when they use it in incorrect ways.

Discontinued

The metadata name has received wide peer review and it has been found wanting. Existing pages are using this metadata name, but new pages should avoid it. The "brief description" and "specification" entries will give details of what authors should use instead, if anything.

If a metadata name is found to be redundant with existing values, it should be removed and listed as a synonym for the existing value.

If a metadata name is registered in the "proposed" state for a period of a month or more without being used or specified, then it may be removed from the registry.

If a metadata name is added with the "proposed" status and found to be redundant with existing values, it should be removed and listed as a synonym for the existing value. If a metadata name is added with the "proposed" status and found to be harmful, then it should be changed to "discontinued" status.

Anyone can change the status at any time, but should only do so in accordance with the definitions above.

Conformance checkers may use the information given on the WHATWG Wiki MetaExtensions page to establish if a value is allowed or not: values defined in this specification or marked as "proposed" or "ratified" must be accepted, whereas values marked as "discontinued" or not listed in either this specification or on the aforementioned page must be reported as invalid. Conformance checkers may cache this information (e.g., for performance reasons or to avoid the use of unreliable network connectivity).

When an author uses a new metadata name not defined by either this specification or the Wiki page, conformance checkers should offer to add the value to the Wiki, with the details described above, with the "proposed" status.

Metadata names whose values are to be [URLs](#) must not be proposed or accepted. Links must be represented using the `<link>` element, not the `<meta>` element.

4.2.5.3. *Pragma directives*

When the **http-equiv** attribute is specified on a `<meta>` element, the element is a pragma directive.

The http-equiv attribute is an [enumerated attribute](#). The following table lists the keywords defined for this attribute. The states given in the first cell of the rows with keywords give the states to which those keywords map. Some of the keywords are non-conforming, as noted in the last column.

State	Keyword	Notes
Content Language	content-language	Non-conforming
Encoding declaration	content-type	
Default style	default-style	
Refresh	refresh	
Cookie setter	set-cookie	Non-conforming

When a `<meta>` element is [inserted into the document](#), if its `http-equiv` attribute is present and represents one of the above states, then the user agent must run the algorithm appropriate for that state, as described in the following list:

Content language state (`http-equiv="content-language"`)

NOTE:

This feature is non-conforming. Authors are encouraged to use the [lang](#) attribute instead.

This pragma sets the **pragma-set default language**. Until such a pragma is successfully processed, there is no [pragma-set default language](#).

1. If the `<meta>` element has no `content` attribute, then abort these steps.
2. If the element's `content` attribute contains a U+002C COMMA character (,) then abort these steps.
3. Let *input* be the value of the element's `content` attribute.
4. Let *position* point at the first character of *input*.
5. [Skip whitespace](#).
6. [Collect a sequence of characters](#) that are not [space characters](#).
7. Let *candidate* be the string that resulted from the previous step.
8. If *candidate* is the empty string, abort these steps.
9. Set the [pragma-set default language](#) to *candidate*.

NOTE:

If the value consists of multiple space-separated tokens, tokens after the first are ignored.

NOTE:

This pragma is not the same as the HTTP Content-Language header of the same name. HTTP Content-Language values with more than one language tag will be rejected as invalid by this pragma. [\[HTTP\]](#)

Encoding declaration state (`http-equiv="content-type"`)

The [encoding declaration state](#) is just an alternative form of setting the `charset` attribute: it is a [character encoding declaration](#). This state's user agent requirements are all handled by the parsing section of the specification.

For `<meta>` elements with an `http-equiv` attribute in the [encoding declaration state](#), the `content` attribute must have a value that is an [ASCII case-insensitive](#) match for a string that consists of: the literal string "text/html;", optionally followed by any number of [space characters](#), followed by the literal string "charset=", followed by one of the [labels](#) of the [character encoding](#) of the [character encoding declaration](#).

A document must not contain both a `<meta>` element with an `http-equiv` attribute in the [encoding declaration state](#) and a `<meta>` element with the `charset` attribute present.

The [encoding declaration state](#) may be used in [HTML documents](#) and in [XML Documents](#). If the [encoding declaration state](#) is used in [XML Documents](#), the name of the [character encoding](#) must be an [ASCII case-insensitive](#) match for the string "UTF-8" (and the document is therefore forced to use UTF-8 as its encoding).

NOTE:

The [encoding declaration state](#) has no effect in XML documents, and is only allowed in order to facilitate migration to and from XHTML.

Default style state (`http-equiv="default-style"`)

This pragma sets the name of the default [alternative style sheet set](#).

1. If the `<meta>` element has no `content` attribute, or if that attribute's value is the empty string, then abort these steps.
2. Set the [preferred style sheet set](#) to the value of the element's `content` attribute. [\[CSSOM\]](#)

Refresh state (`http-equiv="refresh"`)

This pragma acts as timed redirect.

1. If another `<meta>` element with an `http-equiv` attribute in the [Refresh state](#) has already been successfully processed (i.e., when it was inserted the user agent processed it and reached the step labeled *end*), then abort these steps.
2. If the `<meta>` element has no `content` attribute, or if that attribute's value is the empty string, then abort these steps.
3. Let *input* be the value of the element's `content` attribute.
4. Let *position* point at the first character of *input*.
5. [Skip whitespace](#).
6. [Collect a sequence of characters](#) that are [ASCII digits](#), and parse the resulting string using the [rules for parsing non-negative integers](#). If the sequence of characters collected is the empty string, then no number will have been parsed; abort these steps. Otherwise, let *time* be the parsed number.
7. [Collect a sequence of characters](#) that are [ASCII digits](#) and U+002E FULL STOP characters (.). Ignore any collected characters.
8. Let *url* be the `<meta>` element's [node document](#)'s [URL](#).
9. If *position* is past the end of *input*, jump to the step labeled *end*.
10. If the character in *input* pointed to by *position* is not a U+003B SEMICOLON character (;), a U+002C COMMA character (,), or a [space character](#), then abort these steps.
11. [Skip whitespace](#).
12. If the character in *input* pointed to by *position* is a U+003B SEMICOLON character (;), a U+002C COMMA character (,), then advance *position* to the next character.
13. [Skip whitespace](#).
14. If *position* is past the end of *input*, jump to the step labeled *end*.
15. Let *url* be equal to the substring of *input* from the character at *position* to the end of the string.
16. If the character in *input* pointed to by *position* is a U+0055 LATIN CAPITAL LETTER U character (U) or a U+0075 LATIN SMALL LETTER U character (u), then advance *position* to the next character. Otherwise, jump to the step labeled *skip quotes*.
17. If the character in *input* pointed to by *position* is a U+0052 LATIN CAPITAL LETTER R character (R) or a U+0072 LATIN SMALL LETTER R character (r), then advance *position* to the next character. Otherwise, jump to the step labeled *Parse*.
18. If the character in *input* pointed to by *position* is a U+004C LATIN CAPITAL LETTER L character (L) or a U+006C LATIN SMALL LETTER L character (l), then advance *position* to the next character. Otherwise, jump to the step labeled *Parse*.

19. [Skip whitespace](#).
20. If the character in `input` pointed to by `position` is a U+003D EQUALS SIGN (=), then advance `position` to the next character. Otherwise, jump to the step step labeled *Parse*.
21. [Skip whitespace](#).
22. *Skip quotes*: If the character in `input` pointed to by `position` is either a U+0027 APOSTROPHE character (') or U+0022 QUOTATION MARK character ("), then let `quote` be that character, and advance `position` to the next character. Otherwise, let `quote` be the empty string.
23. Let `url` be equal to the substring of `input` from the character at `position` to the end of the string.
24. If `quote` is not the empty string, and there is a character in `url` equal to `quote`, then truncate `url` at that character, so that it and all subsequent characters are removed.
25. *Parse*: *Parse* `url` relative to the `<meta>` element's [node document](#). If that fails, abort these steps. Otherwise, let `urlRecord` be the [resulting URL record](#).
26. *End*: Perform one or more of the following steps:
 - After the refresh has come due (as defined below), if the user has not canceled the redirect and if the `<meta>` element's [node document](#)'s [active sandboxing flag set](#) does not have the [sandboxed automatic features browsing context flag set](#), [navigate the Document's browsing context](#) to `urlRecord`, with [replacement enabled](#), and with the [Document's browsing context](#) as the [source browsing context](#).

For the purposes of the previous paragraph, a refresh is said to have come due as soon as the *later* of the following two conditions occurs:

 - At least *time* seconds have elapsed since the document has [completely loaded](#), adjusted to take into account user or user agent preferences.
 - At least *time* seconds have elapsed since the `<meta>` element was [inserted into the document](#), adjusted to take into account user or user agent preferences.
 - Provide the user with an interface that, when selected, [navigates a browsing context](#) to `urlRecord`, with the [Document's browsing context](#) as the [source browsing context](#).
 - Do nothing.

In addition, the user agent may, as with anything, inform the user of any and all aspects of its operation, including the state of any timers, the destinations of any timed redirects, and so forth.

For `<meta>` elements with an `http-equiv` attribute in the [Refresh state](#), the `content` attribute must have a value consisting either of:

- just a [valid non-negative integer](#), or
- a [valid non-negative integer](#), followed by a U+003B SEMICOLON character (;), followed by one or more [space characters](#), followed by a substring that is an [ASCII case-insensitive](#) match for the string "URL", followed by a U+003D EQUALS SIGN character (=), followed by a [valid URL](#) that does not start with a literal U+0027 APOSTROPHE (') or U+0022 QUOTATION MARK (") character.

In the former case, the integer represents a number of seconds before the page is to be reloaded; in the latter case the integer represents a number of seconds before the page is to be replaced by the page at the given [URL](#).

EXAMPLE 95

A news organization's front page could include the following markup in the page's `<head>` element, to ensure that the page automatically reloads from the server every five minutes:

```
<meta http-equiv="Refresh" content="300">
```

EXAMPLE 96

A sequence of pages could be used as an automated slide show by making each page refresh to the next page in the sequence, using markup such as the following:

```
<meta http-equiv="Refresh" content="20; URL=page4.html">
```

Cookie setter (`http-equiv="set-cookie"`)

This pragma sets an HTTP cookie. [\[COOKIES\]](#)

It is non-conforming. Real HTTP headers should be used instead.

1. If the `<meta>` element has no `content` attribute, or if that attribute's value is the empty string, then abort these steps.
2. Act as if [receiving a set-cookie-string](#) for the document's address via a "non-HTTP" API, consisting of the value of the element's `content` attribute [encoded as UTF-8](#). [\[COOKIES\]](#) [\[ENCODING\]](#)

Content security policy state (`http-equiv="content-security-policy"`)

This pragma [enforces](#) a [Content Security Policy](#) on a [Document](#). [\[CSP3\]](#)

1. If the `<meta>` element is not a child of a `<head>` element, abort these steps.
2. If the `<meta>` element has no `content` attribute, or if that attribute's value is the empty string, then abort these steps.
3. Let *policy* be the result of executing Content Security Policy's [parse a serialized Content Security Policy](#) algorithm on the `<meta>` element's `content` attribute's value.
4. Remove all occurrences of the [report-uri](#), [frame-ancestors](#), and [sandbox directives](#) from *policy*.
5. [Enforce the policy](#) *policy*.

For `<meta>` elements with an `http-equiv` attribute in the [Content security policy state](#), the `content` attribute must have a value consisting of a [valid Content Security Policy](#), but must not contain any [report-uri](#), [frame-ancestors](#), or [sandbox directives](#). The [Content Security Policy](#) given in the `content` attribute will be [enforced](#) upon the current document. [\[CSP3\]](#)

EXAMPLE 97

A page might choose to mitigate the risk of cross-site scripting attacks by preventing the execution of inline JavaScript, as well as blocking all plugin content, using a policy such as the following:

```
<meta http-equiv="Content-Security-Policy" content="script-src 'self'; object-src 'none' ">
```

There must not be more than one `<meta>` element with any particular state in the document at a time.

4.2.5.4. Other pragma directives

Extensions to the predefined set of pragma directives may, under certain conditions, be registered in the [WHATWG Wiki PragmaExtensions page](#). [\[WHATGWWIKI\]](#)

Such extensions must use a name that is identical to an HTTP header registered in the Permanent Message Header Field Registry, and must have behavior identical to that described for the HTTP header. [\[IANAPERMHEDERS\]](#)

Pragma directives corresponding to headers describing metadata, or not requiring specific user agent processing, must not be registered; instead, use [metadata names](#). Pragma directives corresponding to headers that affect the HTTP processing model (e.g., caching) must not be registered, as they would result in HTTP-level behavior being different for user agents that implement HTML than for user agents that do not.

Anyone is free to edit the [WHATWG Wiki PragmaExtensions page](#) at any time to add a pragma directive satisfying these

conditions. Such registrations must specify the following information:

Keyword

The actual name being defined. The name must match a previously-registered HTTP name with the same requirements.

Brief description

A short non-normative description of the purpose of the pragma directive.

Specification

A link to the specification defining the corresponding HTTP header.

Conformance checkers must use the information given on the [WHATWG Wiki PragmaExtensions page](#) to establish if a value is allowed or not: values defined in this specification or listed on the aforementioned page must be accepted, whereas values not listed in either this specification or on the aforementioned page must be rejected as invalid. Conformance checkers may cache this information (e.g., for performance reasons or to avoid the use of unreliable network connectivity).

4.2.5.5. Specifying the document's character encoding

A **character encoding declaration** is a mechanism by which the [character encoding](#) used to store or transmit a document is specified.

The following restrictions apply to [character encoding declarations](#):

- The character encoding name given must be an [ASCII case-insensitive](#) match for one of the [labels](#) of the [character encoding](#) used to serialize the file. [\[ENCODING\]](#)
- The character encoding declaration must be serialized without the use of [character references](#) or character escapes of any kind.
- The element containing the character encoding declaration must be serialized completely within **the first 1024 bytes** of the document.

In addition, due to a number of restrictions on [<meta>](#) elements, there can only be one meta-based character encoding declaration per document.

If an [HTML document](#) does not start with a BOM, and its [encoding](#) is not explicitly given by [Content-Type metadata](#), and the document is not an [iframe srcdoc document](#), then the character encoding used must be an [ASCII-compatible encoding](#), and the encoding must be specified using a meta element with a charset attribute or a [<meta>](#) element with an http-equiv attribute in the [encoding declaration state](#).

NOTE:

A character encoding declaration is required (either in the [Content-Type metadata](#) or explicitly in the file) even if the encoding is US-ASCII, because a character encoding is needed to process non-ASCII characters entered by the user in forms, in URLs generated by scripts, and so forth.

If the document is an [iframe srcdoc document](#), the document must not have a [character encoding declaration](#). (In this case, the source is already decoded, since it is part of the document that contained the [<iframe>](#).)

If an [HTML document](#) contains a [<meta>](#) element with a charset attribute or a [<meta>](#) element with an http-equiv attribute in the [encoding declaration state](#), then the character encoding used must be an [ASCII-compatible encoding](#).

Authors should use [UTF-8](#). Conformance checkers may advise authors against using legacy encodings. [\[ENCODING\]](#)

Authoring tools should default to using [UTF-8](#) for newly-created documents. [\[ENCODING\]](#)

Authors must not use encodings that are not defined in the WHATWG Encoding standard. Additionally, authors should not use [ISO-2022-JP](#). [\[ENCODING\]](#)

NOTE:

Some encodings that are not defined in the WHATWG Encoding standard use bytes in the range 0x20 to 0x7E, inclusive, to encode characters other than the corresponding characters in the range U+0020 to U+007E, inclusive, and represent a potential security vulnerability: A user agent might end up interpreting supposedly benign plain text content as HTML tags and JavaScript.

NOTE:

Using non-UTF-8 encodings can have unexpected results on form submission and URL encodings, which use the [document's character encoding](#) by default.

In XHTML, the XML declaration should be used for inline character encoding information, if necessary.

EXAMPLE 98

In HTML, to declare that the character encoding is UTF-8, the author could include the following markup near the top of the document (in the [<head>](#) element):

```
<meta charset="utf-8">
```

In XML, the XML declaration would be used instead, at the very top of the markup:

```
<?xml version="1.0" encoding="utf-8"?>
```

§ **4.2.6. The style element****Categories:**

[Metadata content](#).

Contexts in which this element can be used:

Where [metadata content](#) is expected.

In a [<noscript>](#) element that is a child of a [<head>](#) element.

Content model:

Depends on the value of the type attribute, but must match requirements described in prose below.

Tag omission in text/html:

Neither tag is omissible.

Content attributes:

[Global attributes](#)

[media](#) — Applicable media

[nonce](#) - Cryptographic nonce used in *Content Security Policy* checks [\[CSP3\]](#)

[type](#) — Type of embedded resource

Also, the [title](#) attribute has special semantics on this element: Alternative style sheet set name.

Allowed [ARIA role attribute](#) values:

None

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

DOM interface:

```
interface HTMLStyleElement : HTMLElement {
    attribute DOMString media;
    attribute DOMString nonce;
    attribute DOMString type;
};
HTMLStyleElement implements LinkStyle;
```

⚠Warning! There are no known native implementations of blocking the `<style>` element based on CSP3 directives. Therefore this feature should not be relied upon.

The `<style>` element allows authors to embed style information in their documents. The `<style>` element is one of several inputs to the styling processing model. The element does not [represent](#) content for the user.

The **type** attribute gives the styling language. If the attribute is present, its value must be a [valid mime type](#) that designates a styling language. The **charset** parameter must not be specified. The default value for the **type** attribute, which is used if the attribute is absent, is "text/css". [\[RFC2318\]](#)

When examining types to determine if they support the language, user agents must not ignore unknown MIME parameters — types with unknown parameters must be assumed to be unsupported. The **charset** parameter must be treated as an unknown parameter for the purpose of comparing [MIME types](#) here.

The **media** attribute says which media the styles apply to. The value must be a [valid media query list](#). The user agent must apply the styles when the **media** attribute's value [matches the environment](#) and the other relevant conditions apply, and must not apply them otherwise.

NOTE:

The styles might be further limited in scope, e.g., in CSS with the use of `@media` blocks. This specification does not override such further restrictions or requirements.

The default, if the **media** attribute is omitted, is "all", meaning that by default styles apply to all media.

NOTE:

A `<style>` element is restricted to appearing in the `<head>` of the document.

The **nonce** attribute represents a cryptographic nonce ("number used once") which can be used by [Content Security Policy](#) to determine whether or not the style specified by an element will be applied to the document. The value is text. [\[CSP3\]](#)

The **title** attribute on `<style>` elements defines [alternative style sheet sets](#). If the `<style>` element has no **title** attribute, then it has no title; the **title** attribute of ancestors does not apply to the `<style>` element. [\[CSSOM\]](#)

NOTE:

The **title** attribute on `<style>` elements, like the **title** attribute on `<link>` elements, differs from the global **title** attribute in that a `style` block without a title does not inherit the title of the parent element: it merely has no title.

The [textContent](#) of a `<style>` element must match the style production in the following ABNF, the character set for which is Unicode. [\[ABNF\]](#)

```
style      = no-c-start *( c-start no-c-end c-end no-c-start )
no-c-start = < any string that doesn't contain a substring that matches c-start >
c-start    = "<!--"
no-c-end   = < any string that doesn't contain a substring that matches c-end >
c-end      = "-->"
```



The user agent must run the **update a style block** algorithm that applies for the style sheet language specified by the `<style>` element's **type** attribute, passing it the element's **style data**, whenever one of the following conditions occur:

- the element is popped off the [stack of open elements](#) of an [HTML parser](#) or [XML parser](#),
- the element is not on the [stack of open elements](#) of an [HTML parser](#) or [XML parser](#), and it is [inserted into a document](#) or [removed from a document](#),
- the element is not on the [stack of open elements](#) of an [HTML parser](#) or [XML parser](#), and one of its child nodes is modified by a script,

For styling languages that consist of pure text (as opposed to XML), a `style` element's [style data](#) is the concatenation of the

contents of all the Text nodes that are children of the [<style>](#) element (not any other nodes such as comments or elements), in [tree order](#). For XML-based styling languages, the [style data](#) consists of all the child nodes of the [<style>](#) element.

The [update a style block](#) algorithm for CSS (text/css) is as follows:

1. Let *element* be the [<style>](#) element.
2. If *element* has an [associated CSS style sheet, remove the CSS style sheet](#) in question.
3. If *element* is not [in a Document](#), then abort these steps.
4. If the [Should element's inline behavior be blocked by Content Security Policy?](#) algorithm returns "Blocked" when executed upon the [<style>](#) element, "style", and the [<style>](#) element's [style data](#), then abort these steps. [\[CSP3\]](#)
5. [create a CSS style sheet](#) with the following properties:

[type](#)

text/css

[owner node](#)

element

[media](#)

The [media](#) attribute of *element*.

NOTE:

This is a reference to the (possibly absent at this time) attribute, rather than a copy of the attribute's current value. The CSSOM specification defines what happens when the attribute is dynamically set, changed, or removed.

[title](#)

The [title](#) attribute of *element*.

NOTE:

Again, this is a reference to the attribute.

[alternate flag](#)

Unset.

[origin-clean flag](#)

Set.

[parent CSS style sheet](#)

[owner CSS rule](#)

null

[disabled flag](#)

Left at its default value.

[CSS rules](#)

Left uninitialized.

This specification does not define any other styling language's *update a style block* algorithm.

Once the attempts to obtain the style sheet's [critical subresources](#), if any, are complete, or, if the style sheet has no [critical subresources](#), once the style sheet has been parsed and processed, the user agent must, if the loads were successful or there were none, [queue a task to fire a simple event](#) named load at the [<style>](#) element, or, if one of the style sheet's [critical subresources](#) failed to completely load for any reason (e.g., DNS error, HTTP 404 response, a connection being prematurely closed, unsupported Content-Type), [queue a task to fire a simple event](#) named error at the [<style>](#) element. Non-network errors in processing the style sheet or its subresources (e.g., CSS parse errors, PNG decoding errors) are not failures for the purposes of this paragraph.

The [task source](#) for these [tasks](#) is the [DOM manipulation task source](#).

The element must [delay the load event](#) of the element's [node document](#) until all the attempts to obtain the style sheet's [critical subresources](#), if any, are complete.

NOTE:

This specification does not specify a style system, but CSS is expected to be supported by most Web browsers. [\[CSS-2015\]](#)

The **media**, **nonce**, and **type** IDL attributes must [reflect](#) the respective content attributes of the same name.

The LinkStyle interface is also implemented by this element. [\[CSSOM\]](#)

EXAMPLE 99

The following document has its stress emphasis styled as bright red text rather than italics text, while leaving titles of works and Latin words in their default italics. It shows how using appropriate elements enables easier restyling of documents.

```
<!DOCTYPE html><html>
  <head>
    <title>My favorite book</title>
    <style>
      body { color: black; background: white; }
      em { font-style: normal; color: red; }
    </style>
  </head>
  <body>
    <p>My <em>favorite</em> book of all time has <em>got</em> to be
    <cite>A Cat's Life</cite>. It is a book by P. Rahmel that talks
    about the <i lang="la">Felis Catus</i> in modern human society.</p>
  </body>
</html>
```

§ 4.2.7. Interactions of styling and scripting

Style sheets, whether added by a [link](#) element, a [style](#) element, an `<?xml-stylesheet?>` PI, an HTTP Link header, or some other mechanism, have a **style sheet ready** flag, which is initially unset.

When a style sheet is ready to be applied, its [style sheet ready](#) flag must be set. If the style sheet referenced no other resources (e.g., it was an internal style sheet given by a [style](#) element with no `@import` rules), then the style rules must be [immediately](#) made available to script; otherwise, the style rules must only be made available to script once the [event loop](#) reaches its *update the rendering* step.

A style sheet in the context of the [Document](#) of an [HTML parser](#) or [XML parser](#) is said to be a **style sheet that is blocking scripts** if the element was created by that [Document](#)'s parser, and the element is either a [style](#) element or a [link](#) element that was an [external resource link](#) when the element was created by the parser, and the element's style sheet was enabled when the element was created by the parser, and the element's [style sheet ready](#) flag is not yet set, and, the last time the [event loop](#) reached step 1, the element was in that [Document](#), and the user agent hasn't given up on that particular style sheet yet. A user agent may give up on a style sheet at any time.

NOTE:

Giving up on a style sheet before the style sheet loads, if the style sheet eventually does still load, means that the script might end up operating with incorrect information. For example, if a style sheet sets the color of an element to green, but a script that inspects the resulting style is executed before the sheet is loaded, the script will find that the element is black (or whatever the default color is), and might thus make poor choices (e.g., deciding to use black as the color elsewhere on the page, instead of green). Implementors have to balance the likelihood of a script using incorrect information with the performance impact of doing nothing while waiting for a slow network request to finish.

A [Document](#) has a **style sheet that is blocking scripts** if there is either a [style sheet that is blocking scripts](#) in the context of that [Document](#), or if that [Document](#) is in a [browsing context](#) that has a [parent browsing context](#), and the [active document](#) of that [parent browsing context](#) itself has a [style sheet that is blocking scripts](#).

A [Document](#) has no style sheet that is blocking scripts if it does not [have a style sheet that is blocking scripts](#) as defined in the previous paragraph.

4.3. Sections

4.3.1. The body element

Categories:

[Sectioning root](#).

Contexts in which this element can be used:

As the second element in an [<html>](#) element.

Content model:

[Flow content](#).

Tag omission in text/html:

A [<body>](#) element's [start tag](#) may be omitted if the element is empty, or if the first thing inside the [<body>](#) element is not a [space character](#) or a [comment](#), except if the first thing inside the [<body>](#) element is a [<meta>](#), [<link>](#), [<script>](#), [<style>](#), or [<template>](#) element.

A [<body>](#) element's [end tag](#) may be omitted if the [<body>](#) element is not immediately followed by a [comment](#).

Content attributes:

[Global attributes](#)

onafterprint

onbeforeprint

onbeforeunload

onhashchange

onlanguagechange

onmessage

onoffline

ononline

onpagehide

onpageshow

onpopstate

onrejectionhandled

onstorage

onunhandledrejection

onunload

Allowed [ARIA role attribute](#) values:

[‘document’](#) role (default - *do not set*), [‘application’](#).

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLBodyElement : HTMLElement {
};
HTMLBodyElement implements WindowEventHandlers;
```

The [<body>](#) element [represents](#) the content of the document.

In [conforming documents](#), there is only one [<body>](#) element. The document.body IDL attribute provides scripts with easy access to a document's [<body>](#) element.

NOTE:

Some DOM operations (for example, parts of the [drag and drop](#) model) are defined in terms of "[the body element](#)". This refers to a particular element in the DOM, as per the definition of the term, and not any arbitrary [<body>](#) element.

The [<body>](#) element exposes as [event handler content attributes](#) a number of the [event handlers](#) of the Window object. It also mirrors their [event handler IDL attributes](#).

The [onblur](#), [onerror](#), [onfocus](#), [onload](#), [onresize](#), and [onscroll](#) [event handlers](#) of the Window object, exposed on the [<body>](#) element, replace the generic [event handlers](#) with the same names normally supported by [html elements](#).

EXAMPLE 100

Thus, for example, a bubbling error event dispatched on a child of [the body element](#) of a [Document](#) would first trigger the [onerror event handler content attributes](#) of that element, then that of the root html element, and only *then* would it trigger the [onerror event handler content attribute](#) on the [<body>](#) element. This is because the event would bubble from the target, to the [<body>](#), to the [<html>](#), to the [Document](#), to the [Window](#), and the [event handler](#) on the [<body>](#) is watching the Window not the [<body>](#). A regular event listener attached to the [<body>](#) using `addEventListener()`, however, would be run when the event bubbled through the body and not when it reaches the Window object.

EXAMPLE 101

This page updates an indicator to show whether or not the user is online:

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Online or offline?</title>
    <script>
      function update(online) {
        document.getElementById('status').textContent =
          online ? 'Online' : 'Offline';
      }
    </script>
  </head>
  <body online="update(true)"
    onoffline="update(false)"
    onload="update(navigator.onLine)">
    <p>You are: <span id="status">(Unknown)</span></p>
  </body>
</html>
```

4.3.2. The article element**Categories:**

[Flow content](#), but with no [<main>](#) element descendants.
[Sectioning content](#).
[Palpable content](#).

Contexts in which this element can be used:

Where [flow content](#) is expected.

Content model:

[Flow content](#).

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[‘article’](#) (default - *do not set*), [‘application’](#), [‘document’](#) or [‘main’](#).

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

[DOM interface](#):

Uses [HTMLElement](#).

The `<article>` element [represents](#) a complete, or self-contained, composition in a document, page, application, or site. This could be a magazine, newspaper, technical or scholarly article, an essay or report, a blog or other social media post.

A general rule is that the `<article>` element is appropriate only if the element's contents would be listed explicitly in the document's outline. Each `<article>` should be identified, typically by including a heading (`<h1>`-`<h6>` element) as a child of the `<article>` element.

Assistive Technology may convey the semantics of the `<article>` to users. This information can provide a hint to users as to the type of content. For example the role of the element, which in this case matches the element name "article", can be announced by screen reader software when a user navigates to an `<article>` element. User Agents may also provide methods to navigate to `<article>` elements.

When `<article>` elements are nested, the inner `<article>` elements represent articles that are in principle related to the contents of the outer article. For instance, a blog entry on a site could consist of summaries of other blog entries in `<article>` elements nested within the `<article>` element for the blog entry.

Author information associated with an `<article>` element (q.v. the address element) does not apply to nested `<article>` elements.

EXAMPLE 102

The following is an example of a blog post extract, marked up using the `<article>` element:

```
<article>
  <header>
    <h2><a href="https://herbert.io">Short note on wearing shorts</a></h2>
    <p>Posted on Wednesday, 10 February 2016 by Patrick Lauke.
    <a href="https://herbert.io/short-note/#comments">6 comments</a></p>
  </header>
  <p>A fellow traveller posed an interesting question: Why do you wear shorts rather than
  longs? The person was wearing culottes as the time, so I considered the question equivocal in
  nature,
  but I attempted to provide an honest answer despite the dubiousness of the questioner's
  dress.</p>
  <p>The short answer is that I enjoy wearing shorts, the long answer is...</p>
  <p><a href="https://herbert.io/short-note/">Continue reading: Short note on
  wearing shorts</a></p>
</article>
```

NOTE:

The schema.org vocabulary can be used to provide more granular information about the type of article, using the [CreativeWork - Article](#) subtypes, other information such as the publication date for the article can also be provided.

EXAMPLE 103

This example shows a blog post using the `article` element, with some `schema.org` annotations:

```
<article itemscope itemtype="https://schema.org/BlogPosting">
  <header>
    <h2 itemprop="headline">The Very First Rule of Life</h2>
    <p><time itemprop="datePublished" datetime="2016-02-28">3 days ago</time></p>
  </header>
  <p>If there's a microphone anywhere near you, assume it's hot and
  sending whatever you're saying to the world. Seriously.</p>
  <p>...</p>
  <footer>
    <a itemprop="discussionUrl" href="?comments=1">Show comments...</a>
  </footer>
</article>
```

Here is that same blog post, but showing some of the comments:

```
<article itemscope itemtype="https://schema.org/BlogPosting"> <header>
<h2 itemprop="headline">The Very First Rule of Life</h2>
<p><time itemprop="datePublished" datetime="2009-10-09">3 days ago</time></p>
</header>
<p>If there's a microphone anywhere near you, assume it's hot and
sending whatever you're saying to the world. Seriously.</p>
<p>...</p>
<section>
  <h3>Comments</h3>
  <ol>
    <li itemprop="comment" itemscope itemtype="https://schema.org/UserComments" id="c1">
      <p>Posted by: <span itemprop="creator" itemscope itemtype="https://schema.org/Person">
        <span itemprop="name">George Washington</span>
      </span></p>
      <p><time itemprop="commentTime" datetime="2009-10-10">15 minutes ago</time></p>
      <p>Yeah! Especially when talking about your lobbyist friends!</p>
    <li itemprop="comment" itemscope itemtype="https://schema.org/UserComments" id="c2">
      <p>Posted by: <span itemprop="creator" itemscope itemtype="https://schema.org/Person">
        <span itemprop="name">George Hammond</span>
      </span></p>
      <p><time itemprop="commentTime" datetime="2009-10-10">5 minutes ago</time></p>
      <p>Hey, you have the same first name as me.</p>
    </li>
  </ol>
</section>
</article>
```

Notice the use of an ordered list `ol` to organize the comments. Also note the comments are a subsection of the article, identified using a section element.

§ 4.3.3. The section element

Categories:

[Flow content.](#)

[Sectioning content.](#)

[Palpable content.](#)

Contexts in which this element can be used:

Where [flow content](#) is expected.

Content model:

[Flow content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed [ARIA role attribute](#) values:

[‘region’](#) role (default - *do not set*), [‘alert’](#), [‘alertdialog’](#), [‘application’](#), [‘contentinfo’](#), [‘dialog’](#), [‘document’](#), [‘log’](#), [‘main’](#), [‘marquee’](#), [‘presentation’](#), [‘search’](#) or [‘status’](#).

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

Uses [HTMLElement](#).

The `<section>` element [represents](#) a generic section of a document or application. A section, in this context, is a thematic grouping of content. Each section should be identified, typically by including a heading (h1-h6 element) as a child of the `<section>` element.

EXAMPLE 104

Examples of sections would be chapters, the various tabbed pages in a tabbed dialog box, or the numbered sections of a thesis. A Web site’s home page could be split into sections for an introduction, news items, and contact information.

NOTE:

Authors are encouraged to use the `<article>` element instead of the `<section>` element when the content is complete, or self-contained, composition.

NOTE:

The `<section>` element is not a generic container element. When an element is needed only for styling purposes or as a convenience for scripting, authors are encouraged to use the `<div>` element instead. A general rule is that the `<section>` element is appropriate only if the element’s contents would be listed explicitly in the document’s [outline](#).

Assistive Technology may convey the semantics of the `<section>` to users when the element has an explicit label. This information can provide a hint to users as to the type of content. For example the role of the element, which in this case is "region", can be announced by screen reader software when a user navigates to an `<section>` element. User Agents may also provide methods to navigate to `<section>` elements.

EXAMPLE 105

In the following example, we see an article (part of a larger Web page) about apples, containing two short sections.

NOTE:

The `<section>` has an `aria-label` attribute providing a brief description of the contents. Assistive technology may convey the `'region'` role along with the `aria-label` value as a hint to users.

```
<article>
  <header>
    <h2>Apples</h2>
    <p>Tasty, delicious fruit!</p>
  </header>
  <p>The apple is the pomaceous fruit of the apple tree.</p>
  <section aria-label="Red apples.">
    <h3>Red Delicious</h3>
    <p>These bright red apples are the most common found in many
      supermarkets.</p>
  </section>
  <section aria-label="Green apples.">
    <h3>Granny Smith</h3>
    <p>These juicy, green apples make a great filling for
      apple pies.</p>
  </section>
</article>
```

EXAMPLE 106

Here is a graduation program with two sections, one for the list of people graduating, and one for the description of the ceremony. (The markup in this example features an uncommon style sometimes used to minimize the amount of [inter-element whitespace](#).)

```
<!DOCTYPE Html>
<html
  ><head
    ><title
      >Graduation Ceremony Summer 2022</title
    ></head
  ><body
    ><h1
      >Graduation</h1
    ><section
      ><h2
        >Ceremony</h2
      ><p
        >Opening Procession</p
      ><p
        >Speech by Validactorian</p
      ><p
        >Speech by Class President</p
      ><p
        >Presentation of Diplomas</p
      ><p
        >Closing Speech by Headmaster</p
    ></section
    ><section
      ><h2
        >Graduates</h2
      ><ul
        ><li
          >Molly Carpenter</li
        ><li
          >Anastasia Luccio</li
        ><li
          >Ebenezar McCoy</li
        ><li
          >Karrin Murphy</li
        ><li
          >Thomas Raith</li
        ><li
          >Susan Rodriguez</li
      ></ul
    ></section
  ></body
></html>
```

EXAMPLE 107

In this example, a book author has marked up some sections as chapters and some as appendices, and uses CSS to style the headers in these two classes of section differently. The whole book is wrapped in an `<article>` element as part of an even larger document containing other books.

```
<style>
  section { border: double medium; margin: 2em; }
  section.chapter h3 { font: 2em Roboto, Helvetica Neue, sans-serif; }
  section.appendix h3 { font: small-caps 2em Roboto, Helvetica Neue, sans-serif; }
</style>
...
<article class="book">
  <header>
    <h2>My Book</h2>
    <p>A sample with not much content</p>
    <p><small>Published by Dummy Publicorp Ltd.</small></p>
  </header>

  <section class="chapter">
    <h3>My First Chapter</h3>
    <p>This is the first of my chapters. It doesn't say much.</p>
    <p>But it has two paragraphs!</p>
  </section>
  <section class="chapter">
    <h3>It Continues: The Second Chapter</h3>
    <p>Bla dee bla, dee bla dee bla. Boom.</p>
  </section>
  <section class="chapter">
    <h3>Chapter Three: A Further Example</h3>
    <p>It's not like a battle between brightness and earthtones would go
    unnoticed.</p>
    <p>But it might ruin my story.</p>
  </section>
  <section class="appendix">
    <h3>Appendix A: Overview of Examples</h3>
    <p>These are demonstrations.</p>
  </section>
  <section class="appendix">
    <h3>Appendix B: Some Closing Remarks</h3>
    <p>Hopefully this long example shows that you <em>can</em> style
    sections, so long as they are used to indicate actual sections.</p>
  </section>
</article>
```

4.3.4. The nav element**Categories:**

[Flow content.](#)
[Sectioning content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [flow content](#) is expected.

Content model:

[Flow content](#), but with no `<main>` element descendants.

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed [ARIA role attribute](#) values:

[‘navigation’](#) role (default - *do not set*) or [‘presentation’](#).

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

Uses [HTMLElement](#).

The [<nav>](#) element [represents](#) a section of a page that links to other pages or to parts within the page: a section with navigation links.

Assistive Technology may convey the semantics of the [<nav>](#) to users. This information can provide a hint to users as to the type of content. For example the role of the element, which in this case is "navigation", can be announced by screen reader software when a user navigates to an [<nav>](#) element. User Agents may also provide methods to navigate to [<nav>](#) elements.

NOTE:

In cases where the content of a [<nav>](#) element represents a list of items, use list markup to aid understanding and navigation.

NOTE:

Not all groups of links on a page need to be in a [<nav>](#) element — the element is primarily intended for sections that consist of major navigation blocks. In particular, it is common for footers to have a short list of links to various pages of a site, such as the terms of service, the home page, and a copyright page. The [<footer>](#) element alone is sufficient for such cases; while a [<nav>](#) element can be used in such cases, it is usually unnecessary.

NOTE:

User agents (such as screen readers) that are targeted at users who can benefit from navigation information being omitted in the initial rendering, or who can benefit from navigation information being immediately available, can use this element as a way to determine what content on the page to initially skip or provide on request (or both).

EXAMPLE 108

In the following example, there are two `<nav>` elements, one for primary navigation around the site, and one for secondary navigation around the page itself.

```
<body>
  <h1>The Wiki Center Of Exampland</h1>
  <nav>
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/events">Current Events</a></li>
      ...more...
    </ul>
  </nav>
  <article>
    <header>
      <h2>Demos in Exampland</h2>
      <p>Written by A. N. Other.</p>
    </header>
    <nav>
      <ul>
        <li><a href="#public">Public demonstrations</a></li>
        <li><a href="#destroy">Demolitions</a></li>
        ...more...
      </ul>
    </nav>
    <div>
      <section id="public">
        <h2>Public demonstrations</h2>
        <p>...more...</p>
      </section>
      <section id="destroy">
        <h2>Demolitions</h2>
        <p>...more...</p>
      </section>
      ...more...
    </div>
    <footer>
      <p><a href="?edit">Edit</a> | <a href="?delete">Delete</a> | <a
href="?Rename">Rename</a></p>
    </footer>
  </article>
  <footer>
    <p><small>© copyright 1998 Exampland Emperor</small></p>
  </footer>
</body>
```

EXAMPLE 109

In the following example, the page has several places where links are present, but only one of those places is considered a navigation section.

```
<body typeof="schema:Blog">
  <header>
    <h1>Wake up sheeple!</h1>
    <p><a href="news.html">News</a> -
    <a href="blog.html">Blog</a> -
    <a href="forums.html">Forums</a></p>
    <p>Last Modified: <span property="schema:dateModified">2009-04-01</span></p>
  <nav>
    <h2>Navigation</h2>
    <ul>
      <li><a href="articles.html">Index of all articles</a></li>
      <li><a href="today.html">Things sheeple need to wake up for today</a></li>
      <li><a href="successes.html">Sheeple we have managed to wake</a></li>
    </ul>
  </nav>
</header>
<main>
  <article property="schema:blogPosts" typeof="schema:BlogPosting">
    <header>
      <h2 property="schema:headline">My Day at the Beach</h2>
    </header>
    <main property="schema:articleBody">
      <p>Today I went to the beach and had a lot of fun.</p>
      ...more content...
    </main>
    <footer>
      <p>Posted <time property="schema:datePublished"
datetime="2009-10-10">Thursday</time>.</p>
    </footer>
  </article>
  ...more blog posts...
</main>
<footer>
  <p>Copyright ©
    <span property="schema:copyrightYear">2010</span>
    <span property="schema:copyrightHolder">The Example Company</span>
  </p>
  <p><a href="about.html">About</a> -
    <a href="policy.html">Privacy Policy</a> -
    <a href="contact.html">Contact Us</a></p>
</footer>
</body>
```

Notice the `<main>` element being used to wrap the main content of the page. In this case, all content other than the page header and footer.

You can also see microdata annotations in the above example that use the schema.org vocabulary to provide the publication date and other metadata about the blog post.

EXAMPLE 110

A `<nav>` element doesn't have to contain a list, it can contain other kinds of content as well. In this navigation block, links are provided in prose:

```
<nav>
  <h2>Navigation</h2>
  <p>You are on my home page. To the north lies <a href="/blog">my
blog</a>, from whence the sounds of battle can be heard. To the east
you can see a large mountain, upon which many <a
href="/school">school papers</a> are littered. Far up this mountain
you can spy a little figure who appears to be me, desperately
scribbling a <a href="/school/thesis">thesis</a>.</p>
  <p>To the west are several exits. One fun-looking exit is labeled <a
href="https://games.example.com/">"games"</a>. Another more
boring-looking exit is labeled <a
href="https://isp.example.net/">ISP</a>.</p>
  <p>To the south lies a dark and dank <a href="/about">contacts
page</a>. Cobwebs cover its disused entrance, and at one point you
see a rat run quickly out of the page.</p>
</nav>
```

EXAMPLE 111

In this example, `<nav>` is used in an e-mail application, to let the user switch folders:

```
<p><input type="button" value="Compose" onclick="compose()"></p>
<nav>
  <h2>Folders</h2>
  <ul>
    <li> <a href="/inbox" onclick="return openFolder(this.href)">Inbox</a> <span class="count">
</span>
    <li> <a href="/sent" onclick="return openFolder(this.href)">Sent</a>
    <li> <a href="/drafts" onclick="return openFolder(this.href)">Drafts</a>
    <li> <a href="/trash" onclick="return openFolder(this.href)">Trash</a>
    <li> <a href="/customers" onclick="return openFolder(this.href)">Customers</a>
  </ul>
</nav>
```

4.3.5. The aside element**Categories:**

[Flow content.](#)
[Sectioning content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [flow content](#) is expected.

Content model:

[Flow content](#), but with no `<main>` element descendants.

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

`'complementary'` role (default - *do not set*), `'note'`, `'search'` or `'presentation'`.

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

Uses [HTMLElement](#).

The `<aside>` element [represents](#) a section of a page that consists of content that is tangentially related to the content of the parenting [sectioning content](#), and which could be considered separate from that content. Such sections are often represented as sidebars in printed typography.

The element can be used for typographical effects like pull quotes or sidebars, for advertising, for groups of [<nav>](#) elements, and for other content that is considered separate from the main content of the nearest ancestor [sectioning content](#).

Assistive Technology may convey the semantics of the `<aside>` to users. This information can provide a hint to users as to the type of content. For example the role of the element, which in this case is "complementary", can be announced by screen reader software when a user navigates to an `<aside>` element. User Agents may also provide methods to navigate to `<aside>` elements.

NOTE:

It's not appropriate to use the `<aside>` element just for parentheticals, since those are part of the main flow of the document.

EXAMPLE 112

The following example shows how an aside is used to mark up background material on Switzerland in a much longer news story on Europe.

```
<aside>
  <h2>Switzerland</h2>
  <p>Switzerland, a land-locked country in the middle of geographic
    Europe, has not joined the geopolitical European Union, though it is
    a signatory to a number of European treaties.</p>
</aside>
```

EXAMPLE 113

The following example shows how an aside is used to mark up a pull quote in a longer article.

```
...

<p>He later joined a large company, continuing on the same work.
<q>I love my job. People ask me what I do for fun when I'm not at
work. But I'm paid to do my hobby, so I never know what to
answer. Some people wonder what they would do if they didn't have to
work... but I know what I would do, because I was unemployed for a
year, and I filled that time doing exactly what I do now.</q></p>

<aside>
  <q> People ask me what I do for fun when I'm not at work. But I'm
  paid to do my hobby, so I never know what to answer. </q>
</aside>

<p>Of course his work – or should that be hobby? –      isn't his only passion. He also enjoys
other pleasures.</p>

...
```

EXAMPLE 114

The following extract shows how `<aside>` can be used for blogrolls and other side content on a blog:

```

<body>
  <header>
    <h1>My wonderful blog</h1>
    <p>My tagline</p>
  </header>
  <aside>
    <!-- this aside contains two sections that are tangentially related
    to the page, namely, links to other blogs, and links to blog posts
    from this blog -->
    <nav>
      <h2>My blogroll</h2>
      <ul>
        <li><a href="https://blog.example.com/">Example Blog</a>
      </ul>
    </nav>
    <nav>
      <h2>Archives</h2>
      <ol reversed>
        <li><a href="/last-post">My last post</a>
        <li><a href="/first-post">My first post</a>
      </ol>
    </nav>
  </aside>
  <aside>
    <!-- this aside is tangentially related to the page also, it
    contains twitter messages from the blog author -->
    <h2>Twitter Feed</h2>
    <blockquote cite="https://twitter.example.net/t31351234">
      I'm on vacation, writing my blog.
    </blockquote>
    <blockquote cite="https://twitter.example.net/t31219752">
      I'm going to go on vacation soon.
    </blockquote>
  </aside>
  <article>
    <!-- this is a blog post -->
    <h2>My last post</h2>
    <p>This is my last post.</p>
    <footer>
      <p><a href="/last-post" rel=bookmark>Permalink</a>
    </footer>
  </article>
  <article>
    <!-- this is also a blog post -->
    <h2>My first post</h2>
    <p>This is my first post.</p>
    <aside>
      <!-- this aside is about the blog post, since it's inside the
      <article> element; it would be wrong, for instance, to put the
      blogroll here, since the blogroll isn't really related to this post
      specifically, only to the page as a whole -->
      <h1>Posting</h1>
      <p>While I'm thinking about it, I wanted to say something about
      posting. Posting is fun!</p>
    </aside>
    <footer>
      <p><a href="/first-post" rel=bookmark>Permalink</a>
    </footer>
  </article>
  <footer>
    <nav>
      <a href="/archives">Archives</a> -      <a href="/about">About me</a> -      <a
href="/copyright">Copyright</a>

```

```

    </nav>
  </footer>
</body>

```

§ 4.3.6. The h1, h2, h3, h4, h5, and h6 elements

Categories:

[Flow content.](#)
[Heading content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [flow content](#) is expected.

Content model:

[Phrasing content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[‘heading’](#) role (default - *do not set*), [‘tab’](#) or [‘presentation’](#).

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLHeadingElement : HTMLElement {};
```

These elements [represent](#) headings for their sections.

These elements have a **rank** given by the number in their name. The h1 element has the highest rank, the h6 element has the lowest rank, and two elements with the same name have equal rank.

Use the [rank](#) of heading elements to create the document outline.

EXAMPLE 115

The following code shows how to mark up a document outline with six levels of headings.

```

<body>
<h1>top level heading</h1>
<section><h2>2nd level heading</h2>
<section><h3>3nd level heading</h3>
  <section><h4>4th level heading</h4>
    <section><h5>5th level heading</h5>
      <section><h6>6th level heading</h6>
    </section>
  </section>
</section>
</section>
</section>
</body>

```

NOTE:

The document outline would be the same if the [<section>](#) elements were not used.

h2–h6 elements must not be used to markup subheadings, subtitles, alternative titles and taglines unless intended to be the heading for a new section or subsection. Instead use the markup patterns in the [§4.13 Common idioms without dedicated elements](#) section of the specification.

Assistive technology often announces the presence and level of a heading to users, as a hint to understand the structure of a document and construct a 'mental model' of its outline. For example the role of the element, which in this case is "heading" and the heading level "1" to "6", can be announced by screen reader software when a user navigates to an h1–h6 element. User Agents may also provide methods to navigate to h1–h6 elements.

EXAMPLE 116

As far as their respective document outlines (their heading and section structures) are concerned, these two snippets are semantically equivalent:

```
<body>
  <h1>Let's call it a draw(ing surface)</h1>
  <h2>Diving in</h2>
  <h2>Simple shapes</h2>
  <h2>Canvas coordinates</h2>
  <h3>Canvas coordinates diagram</h3>
  <h2>Paths</h2>
</body>
```

```
<body>
  <h1>Let's call it a draw(ing surface)</h1>
  <section>
    <h2>Diving in</h2>
  </section>
  <section>
    <h2>Simple shapes</h2>
  </section>
  <section>
    <h2>Canvas coordinates</h2>
    <section>
      <h3>Canvas coordinates diagram</h3>
    </section>
  </section>
  <section>
    <h2>Paths</h2>
  </section>
</body>
```

Authors might prefer the former style for its terseness, or the latter style for its convenience in the face of heavy editing; which is best is purely an issue of preferred authoring style.

The two styles can be combined, for compatibility with legacy tools while still future-proofing for when that compatibility is no longer needed.

NOTE:

The semantics and meaning of the h1–h6 elements are further detailed in the section on [§4.3.10 Headings and sections](#).

§ 4.3.7. The header element

Categories:

[Flow content.](#)

[Palpable content.](#)

Contexts in which this element can be used:

Where [flow content](#) is expected.

Content model:

[Flow content](#), but with no [<main>](#) element descendants, or [<header>](#), [<footer>](#) elements that are not descendants of [sectioning content](#) which is a descendant of the [<header>](#).

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[‘banner’](#) role (default - *do not set*) or [‘presentation’](#).

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

Uses [HTMLElement](#).

The [<header>](#) element [represents](#) introductory content for its nearest ancestor [sectioning content](#) or [sectioning root](#) element. A header typically contains a group of introductory or navigational aids.

When the nearest ancestor [sectioning content](#) or [sectioning root](#) element is [the body element](#), then it applies to the whole page.

NOTE:

A [<header>](#) element is intended to usually contain the section’s heading (an h1–h6 element), but this is not required. The header element can also be used to wrap a section’s table of contents, a search form, or any relevant logos.

Assistive Technology may convey the semantics of the [<header>](#) to users when the nearest ancestor [sectioning content](#) or [sectioning root](#) element is the [<body>](#) element. This information can provide a hint to users as to the type of content. For example the role of the element, which in this case is "banner", can be announced by screen reader software when a user navigates to an [<header>](#) element that is scoped to the [<body>](#) element. User Agents may also provide methods to navigate to [<header>](#) elements scoped to the [<body>](#) element.

EXAMPLE 117

Here are some sample headers. This first one is for a game:

```
<header>
  <p>Welcome to...</p>
  <h1>Voidwars!</h1>
</header>
```

The following snippet shows how the element can be used to mark up a specification's header:

```
<header>
  <h1>Scalable Vector Graphics (SVG) 1.2</h1>
  <p>W3C Working Draft 27 October 2004</p>
  <dl>
    <dt>This version:</dt>
    <dd><a href="https://www.w3.org/TR/2004/WD-SVG12-20041027/">https://www.w3.org/TR/2004/WD-SVG12-20041027/</a></dd>
    <dt>Previous version:</dt>
    <dd><a href="https://www.w3.org/TR/2004/WD-SVG12-20040510/">https://www.w3.org/TR/2004/WD-SVG12-20040510/</a></dd>
    <dt>Latest version of SVG 1.2:</dt>
    <dd><a href="https://www.w3.org/TR/SVG12/">https://www.w3.org/TR/SVG12/</a></dd>
    <dt>Latest SVG Recommendation:</dt>
    <dd><a href="https://www.w3.org/TR/SVG/">https://www.w3.org/TR/SVG/</a></dd>
    <dt>Editor:</dt>
    <dd>Dean Jackson, W3C, <a href="mailto:dean@w3.org">dean@w3.org</a></dd>
    <dt>Authors:</dt>
    <dd>See <a href="#authors">Author List</a></dd>
  </dl>
  <p class="copyright"><a href="https://www.w3.org/Consortium/Legal/ipr-notice">https://www.w3.org/Consortium/Legal/ipr-notice ...</a></p>
</header>
```

NOTE:

The `<header>` element is not [sectioning content](#); it doesn't introduce a new section.

EXAMPLE 118

In this example, the page has a page heading given by the `h1` element, and two subsections whose headings are given by `h2` elements. The content after the `<header>` element is still part of the last subsection started in the `<header>` element, because the `<header>` element doesn't take part in the [outline](#) algorithm.

```
<body>
  <header>
    <h1>Little Green Guys With Guns</h1>
    <nav>
      <ul>
        <li><a href="/games">Games</a>
        <li><a href="/forum">Forum</a>
        <li><a href="/download">Download</a>
      </ul>
    </nav>
    <h2>Important News</h2> <!-- this starts a second subsection -->
    <!-- this is part of the subsection entitled "Important News" -->
    <p>To play today's games you will need to update your client.</p>
    <h2>Games</h2> <!-- this starts a third subsection -->
    </header>
    <p>You have three active games:</p>
    <!-- this is still part of the subsection entitled "Games" -->
    ...
```

NOTE:

For cases where an developer wants to nest a header or footer within another header: The `<header>` element can only contain a `<header>` or `<footer>` if they are themselves contained within [sectioning content](#).

EXAMPLE 119

In this example, the `<article>` has a `<header>` which contains an `aside` which itself contains a `<header>`. This is conforming as the descendant header is contained within the `<aside>` element.

```
<article>
  <header>
    <h1>Flexbox: The definitive guide</h1>
    <aside>
      <header>
        <h2>About the author: Wes McSilly</h2>
        <p><a href="/w3-wes-mcsilly/">Contact him! (Why would you?)</a></p>
      </header>
      <p>Expert in nothing but Flexbox. Talented circus sideshow.</p>
    </aside>
  </header>
  <p><ins>The guide about Flexbox was supposed to be here, but it
    turned out Wes wasn't a Flexbox expert either.</ins></p>
</article>
```

4.3.8. The footer element**Categories:**

[Flow content](#).

[Palpable content](#).

Contexts in which this element can be used:

Where [flow content](#) is expected.

Content model:

[Flow content](#), but with no `<main>` element descendants, or `<header>`, `<footer>` elements that are not descendants of [sectioning content](#) which is a descendant of the `<footer>`.

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

'[contentinfo](#)' role (default - *do not set*) or '[presentation](#)'.

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

Uses [HTMLElement](#).

The `<footer>` element [represents](#) a footer for its nearest ancestor [sectioning content](#) or [sectioning root](#) element. A footer typically contains information about its section such as who wrote it, links to related documents, copyright data, and the like.

When the `<footer>` element contains entire sections, they [represent](#) appendices, indexes, long colophons, verbose license agreements, and other such content.

Assistive Technology may convey the semantics of the `<footer>` to users when the nearest ancestor [sectioning content](#) or [sectioning root](#) element is the `<body>` element. This information can provide a hint to users as to the type of content. For ex-

ample the role of the element, which in this case is "content information", can be announced by screen reader software when a user navigates to an `<footer>` element that is scoped to the `<body>` element. User Agents may also provide methods to navigate to `<footer>` elements scoped to the `<body>` element.

NOTE:

Contact information for the author or editor of a section belongs in an `address` element, possibly itself inside a `<footer>`. Bylines and other information that could be suitable for both a header or a footer can be placed in either (or neither). The primary purpose of these elements is merely to help the author write self-explanatory markup that is easy to maintain and style; they are not intended to impose specific structures on authors.

Footers don't necessarily have to appear at the *end* of a section, though they usually do.

When the nearest ancestor [sectioning content](#) or [sectioning root](#) element is [the body element](#), then it applies to the whole page.

NOTE:

The `<footer>` element is not [sectioning content](#); it doesn't introduce a new section.

EXAMPLE 120

Here is a page with two footers, one at the top and one at the bottom, with the same content:

```
<body>
  <footer><a href="..">Back to index...</a></footer>
  <div>
    <h1>Lorem ipsum</h1>
    <p>The ipsum of all lorem</p>
  </div>
  <p>A dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex
ea commodo consequat. Duis aute irure dolor in reprehenderit in
voluptate velit esse cillum dolore eu fugiat nulla
pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
culpa qui officia deserunt mollit anim id est laborum.</p>
  <footer><a href="..">Back to index...</a></footer>
</body>
```

EXAMPLE 121

Here is an example which shows the `<footer>` element being used both for a site-wide footer and for a section footer.

```
<!DOCTYPE HTML>
<HTML><HEAD>
<TITLE>The Ramblings of a Scientist</TITLE>
<BODY>
<h1>The Ramblings of a Scientist</h1>
<MAIN>
  <ARTICLE>
    <H2>Episode 15</H2>
    <VIDEO SRC="/fm/015.ogv" CONTROLS PRELOAD>
    <P><A HREF="/fm/015.ogv">Download video</A>.</P>
    </VIDEO>
    <FOOTER> <!-- footer for article -->
    <P>Published <TIME DATETIME="2009-10-21T18:26-07:00">on 2009/10/21 at 6:26pm</TIME></P>
    </FOOTER>
  </ARTICLE>
  <ARTICLE>
    <H2>My Favorite Trains</H2>
    <P>I love my trains. My favorite train of all time is a Köf.</P>
    <P>It is fun to see them pull some coal cars because they look so
    dwarfed in comparison.</P>
    <FOOTER> <!-- footer for article -->
    <P>Published <TIME DATETIME="2009-09-15T14:54-07:00">on 2009/09/15 at 2:54pm</TIME></P>
    </FOOTER>
  </ARTICLE>
</MAIN>
<FOOTER> <!-- site wide footer -->
  <NAV>
    <P><A HREF="/credits.html">Credits</A> –      <A HREF="/tos.html">Terms of Service</A> –
    <A HREF="/index.html">Blog Index</A></P>
  </NAV>
  <P>Copyright © 2009 Gordon Freeman</P>
</FOOTER>
</BODY>
</HTML>
```

EXAMPLE 122

Some site designs have what is sometimes referred to as "fat footers" — footers that contain a lot of material, including images, links to other articles, links to pages for sending feedback, special offers... in some ways, a whole "front page" in the footer.

This fragment shows the bottom of a page on a site with a "fat footer":

```
...
<footer>
  <nav>
    <section>
      <h2>Articles</h2>
      <p> Go to the gym with
our somersaults class! Our teacher Jim takes you through the paces
in this two-part article. <a href="articles/somersaults/1">Part
1</a> · <a href="articles/somersaults/2">Part 2</a></p>
      <p> Tired of walking on the edge of
a cliff!-- sic --? Our guest writer Lara shows you how to bumble
your way through the bars. <a href="articles/kindplus/1">Read
more...</a></p>
      <p> The chips are down, now all
that's left is a potato. What can you do with it? <a
href="articles/crisps/1">Read more...</a></p>
    </section>
    <ul>
      <li> <a href="/about">About us...</a>
      <li> <a href="/feedback">Send feedback!</a>
      <li> <a href="/sitemap">Sitemap</a>
    </ul>
  </nav>
  <p><small>Copyright © 2015 The Snacker — <a href="/tos">Terms of Service</a>
</small></p>
</footer>
</body>
```

§ 4.3.9. The address element**Categories:**

[Flow content.](#)

[Palpable content.](#)

Contexts in which this element can be used:

Where [flow content](#) is expected.

Content model:

[Flow content](#), but with no [heading content](#) descendants, no [sectioning content](#) descendants, and no [header](#), [footer](#), or [address](#) element descendants.

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[‘contentinfo’](#) role.

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

Uses [HTML`Element`](#).

The `<address>` element [represents](#) the contact information for its nearest `<article>` or `<body>` element ancestor. If that is [the body element](#), then the contact information applies to the document as a whole.

EXAMPLE 123

For example, a page at the W3C Web site related to HTML might include the following contact information:

```
<ADDRESS>
  <A href="../People/Raggett/">Dave Raggett</A>,
  <A href="../People/Arnaud/">Arnaud Le Hors</A>,
  contact persons for the <A href="Activity">W3C HTML Activity</A>
</ADDRESS>
```

The `<address>` element must not be used to represent arbitrary addresses (e.g., postal addresses), unless those addresses are in fact the relevant contact information. (The `<p>` element is the appropriate element for marking up postal addresses in general.)

The `<address>` element must not contain information other than contact information.

EXAMPLE 124

For example, the following is non-conforming use of the `<address>` element:

```
<ADDRESS>Last Modified: 1999/12/24 23:37:50</ADDRESS>
```

Typically, the `<address>` element would be included along with other information in a `<footer>` element.

The contact information for a node *node* is a collection of address elements defined by the first applicable entry from the following list:

↪ If *node* is an `<article>` element

↪ If *node* is a `<body>` element

The contact information consists of all the `<address>` elements that have *node* as an ancestor and do not have another body or `<article>` element ancestor that is a descendant of *node*.

↪ If *node* has an ancestor element that is an `<article>` element

↪ If *node* has an ancestor element that is a `<body>` element

The contact information of *node* is the same as the contact information of the nearest `<article>` or `<body>` element ancestor, whichever is nearest.

↪ If *node*'s [node document](#) has a `<body>` element

The contact information of *node* is the same as the contact information of [the body element](#) of the [Document](#).

↪ Otherwise

There is no contact information for *node*.

User agents may expose the contact information of a node to the user, or use it for other purposes, such as indexing sections based on the sections' contact information.

EXAMPLE 125

In this example the footer contains contact information and a copyright notice.

```
<footer>
  <address>
    For more details, contact
    <a href="mailto:js@example.com">John Smith</a>.
  </address>
  <p><small>© copyright 2038 Example Corp.</small></p>
</footer>
```

§ 4.3.10. Headings and sections

The h1–h6 elements are headings.

The first element of [heading content](#) in an element of [sectioning content](#) [represents](#) the heading for that section. Subsequent headings of equal or higher [rank](#) start new (implied) sections, headings of lower [rank](#) start implied subsections that are part of the previous one. In both cases, the element [represents](#) the heading of the implied section.

h1–h6 elements must not be used to markup subheadings, subtitles, alternative titles and taglines unless intended to be the heading for a new section or subsection. Instead use the markup patterns in the [§4.13 Common idioms without dedicated elements](#) section of the specification.

Certain elements are said to be **sectioning roots**, including `blockquote` and `<td>` elements. These elements can have their own outlines, but the sections and headings inside these elements do not contribute to the outlines of their ancestors.

⇒ [<blockquote>](#), [<body>](#), [<details>](#), [<fieldset>](#), [<figure>](#), [<td>](#)

[Sectioning content](#) elements are always considered subsections of their nearest ancestor [sectioning root](#) or their nearest ancestor element of [sectioning content](#), whichever is nearest, regardless of what implied sections other headings may have created.

EXAMPLE 126

For the following fragment:

```
<body>
  <h1>Foo</h1>
  <h2>Bar</h2>
  <blockquote>
    <h3>B1a</h3>
  </blockquote>
  <p>Baz</p>
  <h2>Quux</h2>
  <section>
    <h3>Thud</h3>
  </section>
  <p>Grunt</p>
</body>
```

...the structure would be:

1. Foo (heading of explicit body section, containing the "Grunt" paragraph)
 1. Bar (heading starting implied section, containing a block quote and the "Baz" paragraph)
 2. Quux (heading starting implied section with no content other than the heading itself)
 3. Thud (heading of explicit section section)

Notice how the `section` ends the earlier implicit section so that a later paragraph ("Grunt") is back at the top level.

Sections may contain headings of a [rank](#) equal to their section nesting level. Authors should use headings of the appropriate [rank](#) for the section's nesting level.

Authors are also encouraged to explicitly wrap sections in elements of [sectioning content](#), instead of relying on the implicit sections generated by having multiple headings in one element of [sectioning content](#).

EXAMPLE 127

For example, the following is correct:

```
<body>
  <h1>Apples</h1>
  <p>Apples are fruit.</p>
  <section>
    <h2>Taste</h2>
    <p>They taste lovely.</p>
    <h3>Sweet</h3>
    <p>Red apples are sweeter than green ones.</p>
    <h3>Color</h3>
    <p>Apples come in various colors.</p>
  </section>
</body>
```

However, the same document would be more clearly expressed as:

```
<body>
  <h1>Apples</h1>
  <p>Apples are fruit.</p>
  <section>
    <h2>Taste</h2>
    <p>They taste lovely.</p>
    <section>
      <h3>Sweet</h3>
      <p>Red apples are sweeter than green ones.</p>
    </section>
    <section>
      <h3>Color</h3>
      <p>Apples come in various colors.</p>
    </section>
  </section>
</body>
```

Both of the documents above are semantically identical and would produce the same outline in compliant user agents.

4.3.10.1. Creating an outline

⚠Warning! There are currently no known native implementations of the outline algorithm in graphical browsers or assistive technology user agents, although the algorithm is implemented in other software such as conformance checkers and browser extensions. Therefore the [outline](#) algorithm cannot be relied upon to convey document structure to users. Authors should use heading [rank](#) (h1-h6) to convey document structure.

This section is non-normative

This section defines an algorithm for creating an outline for a [sectioning content](#) element or a [sectioning root](#) element. It is defined in terms of a walk over the nodes of a DOM tree, in [tree order](#), with each node being visited when it is *entered* and when it is *exited* during the walk.

The **outline** for a [sectioning content](#) element or a [sectioning root](#) element consists of a list of one or more potentially nested [sections](#). The element for which an [outline](#) is created is said to be **the outline's owner**.

A **section** is a container that corresponds to some nodes in the original DOM tree. Each section can have one heading associated with it, and can contain any number of further nested sections. The algorithm for the outline also associates each node in the DOM tree with a particular section and potentially a heading. (The sections in the outline aren't [<section>](#) elements, though some may correspond to such elements — they are merely conceptual sections.)

EXAMPLE 128

The following markup fragment:

```
<body>
  <h1>A</h1>
  <p>B</p>
  <h2>C</h2>
  <p>D</p>
  <h2>E</h2>
  <p>F</p>
</body>
```

...results in the following outline being created for the body node (and thus the entire document):

1. Section created for body node. Associated with heading "A". Also associated with paragraph "B". Nested sections:
 1. Section implied for first h2 element. Associated with heading "C". Also associated with paragraph "D". No nested sections.
 2. Section implied for second h2 element. Associated with heading "E". Also associated with paragraph "F". No nested sections.

The algorithm that must be followed during a walk of a DOM subtree rooted at a [sectioning content](#) element or a [sectioning root](#) element to determine that element's [outline](#) is as follows:

1. Let *current outline target* be null. (It holds the element whose [outline](#) is being created.)
2. Let *current section* be null. (It holds a pointer to a [section](#), so that elements in the DOM can all be associated with a section.)
3. Create a stack to hold elements, which is used to handle nesting. Initialize this stack to empty.
4. Walk over the DOM in [tree order](#), starting with the [sectioning content](#) element or [sectioning root](#) element at the root of the subtree for which an outline is to be created, and trigger the first relevant step below for each element as the walk enters and exits it.

↪ **When exiting an element, if that element is the element at the top of the stack**

NOTE:

The element being exited is a [heading content](#) element or an element with a hidden attribute.

Pop that element from the stack.

↪ **If the top of the stack is a [heading content](#) element or an element with a hidden attribute**

Do nothing.

↪ **When entering an element with a hidden attribute**

Push the element being entered onto the stack. (This causes the algorithm to skip that element and any descendants of the element.)

↪ **When entering a [sectioning content](#) element**

Run these steps:

1. If *current outline target* is not null, run these substeps:
 1. If the *current section* has no heading, create an implied heading and let that be the heading for the *current section*.
 2. Push *current outline target* onto the stack.

2. Let *current outline target* be the element that is being entered.
3. Let *current section* be a newly created [section](#) for the *current outline target* element.
4. Associate *current outline target* with *current section*.
5. Let there be a new [outline](#) for the new *current outline target*, initialized with just the new *current section* as the only [section](#) in the outline.

↪ When exiting a [sectioning content](#) element, if the stack is not empty

Run these steps:

1. If the *current section* has no heading, create an implied heading and let that be the heading for the *current section*.
2. Pop the top element from the stack, and let the *current outline target* be that element.
3. Let *current section* be the last section in the [outline](#) of the *current outline target* element.
4. Append the [outline](#) of the [sectioning content](#) element being exited to the *current section*. (This does not change which section is the last section in the [outline](#).)

↪ When entering a [sectioning root](#) element

Run these steps:

1. If *current outline target* is not null, push *current outline target* onto the stack.
2. Let *current outline target* be the element that is being entered.
3. Let *current outline target*'s *parent section* be *current section*.
4. Let *current section* be a newly created [section](#) for the *current outline target* element.
5. Let there be a new [outline](#) for the new *current outline target*, initialized with just the new *current section* as the only [section](#) in the outline.

↪ When exiting a [sectioning root](#) element, if the stack is not empty

Run these steps:

1. If the *current section* has no heading, create an implied heading and let that be the heading for the *current section*.
2. Let *current section* be *current outline target*'s *parent section*.
3. Pop the top element from the stack, and let the *current outline target* be that element.

↪ When exiting a [sectioning content](#) element or a [sectioning root](#) element (when the stack is empty)

NOTE:

The *current outline target* is the element being exited, and it is the [sectioning content](#) element or a [sectioning root](#) element at the root of the subtree for which an outline is being generated.

If the *current section* has no heading, create an implied heading and let that be the heading for the *current section*.

Skip to the next step in the overall set of steps. (The walk is over.)

↪ When entering a [heading content](#) element

If the *current section* has no heading, let the element being entered be the heading for the *current section*. Otherwise, if the element being entered has a [rank](#) equal to or higher than the heading of the last section of the [outline](#) of the *current outline target*, or if the heading of the last section of the [outline](#) of the *current outline target* is an implied heading, then create a new [section](#) and append it to the [outline](#) of the *current outline target* element, so that this new section is the new last section of that outline. Let *current section* be that new section. Let the element being entered be the new heading for the *current section*.

Otherwise, run these substeps:

1. Let *candidate section* be *current section*.
2. *Heading loop*: If the element being entered has a [rank](#) lower than the [rank](#) of the heading of the *candidate section*, then create a new [section](#), and append it to *candidate section*. (This does not change which section is the last section in the outline.) Let *current section* be this new section. Let the element being entered be the new heading for the *current section*. Abort these substeps.
3. Let *new candidate section* be the [section](#) that contains *candidate section* in the [outline](#) of *current outline target*.
4. Let *candidate section* be *new candidate section*.
5. Return to the step labeled *heading loop*.

Push the element being entered onto the stack. (This causes the algorithm to skip any descendants of the element.)

NOTE:

Recall that h1 has the highest rank, and h6 has the lowest rank.

↪ **Otherwise**

Do nothing.

In addition, whenever the walk exits a node, after doing the steps above, if the node is not associated with a [section](#) yet, **associate the node with the [section](#) *current section***.

5. Associate all non-element nodes that are in the subtree for which an outline is being created with the [section](#) with which their parent element is associated.
6. Associate all nodes in the subtree with the heading of the [section](#) with which they are associated, if any.

The tree of sections created by the algorithm above, or a proper subset thereof, must be used when generating document outlines, for example when generating tables of contents.

The outline created for [the body element](#) of a [Document](#) is the [outline](#) of the entire document.

When creating an interactive table of contents, entries should jump the user to the relevant [sectioning content](#) element, if the [section](#) was created for a real element in the original document, or to the relevant [heading content](#) element, if the [section](#) in the tree was generated for a heading in the above process.

NOTE:

Selecting the first [section](#) of the document therefore always takes the user to the top of the document, regardless of where the first heading in the body is to be found.

The **outline depth** of a [heading content](#) element associated with a [section](#) *section* is the number of [sections](#) that are ancestors of *section* in the outermost [outline](#) that *section* finds itself in when the [outlines](#) of its [Document](#)'s elements are created, plus 1. The **outline depth** of a [heading content](#) element not associated with a [section](#) is 1.

User agents should provide default headings for sections that do not have explicit section headings.

EXAMPLE 129

Consider the following snippet:

```
<body>
  <nav>
    <p><a href="/">Home</a></p>
  </nav>
  <p>Hello world.</p>
  <aside>
    <p>My cat is cute.</p>
  </aside>
</body>
```

Although it contains no headings, this snippet has three sections: a document (the `<body>`) with two subsections (a `<nav>` and an `<aside>`). A user agent could present the outline as follows:

1. Untitled document
 1. Navigation
 2. Sidebar

These default headings ("Untitled document", "Navigation", "Sidebar") are not specified by this specification, and might vary with the user's language, the page's language, the user's preferences, the user agent implementor's preferences, etc.

The following JavaScript function shows how the tree walk could be implemented. The `root` argument is the root of the tree to walk (either a [sectioning content](#) element or a [sectioning root](#) element), and the `enter` and `exit` arguments are callbacks that are called with the nodes as they are entered and exited. [\[ECMA-262\]](#)

```
function (root, enter, exit) {
  var node = root;
  start: while (node) {
    enter(node);
    if (node.firstChild) {
      node = node.firstChild;
      continue start;
    }
    while (node) {
      exit(node);
      if (node == root) {
        node = null;
      } else if (node.nextSibling) {
        node = node.nextSibling;
        continue start;
      } else {
        node = node.parentNode;
      }
    }
  }
}
```

§ 4.3.11. Usage summary

This section is non-normative.

Element	Purpose
	Example
<code><body></code>	<pre> <!DOCTYPE HTML> <html> <head> <title>Steve Hill's Home Page</title> </head> <body> <p>Hard Trance is My Life.</p> </body> </html> </pre>
<code><article></code>	<pre> <article> <p>My fave Masif tee so far!</p> <footer>Posted 2 days ago</footer> </article> <article> <p>Happy 2nd birthday Masif Saturdays!!!</p> <footer>Posted 3 weeks ago</footer> </article> </pre>
<code><section></code>	<pre> <h1>Biography</h1> <section> <h1>The facts</h1> <p>1500+ shows, 14+ countries</p> </section> <section> <h1>2010/2011 figures per year</h1> <p>100+ shows, 8+ countries</p> </section> </pre>
<code><nav></code>	<pre> <nav> Home Bio Discog </nav> </pre>
<code><aside></code>	<pre> <h1>Music</h1> <p>As any burner can tell you, the event has a lot of trance.</p> <aside>You can buy the music we played at our playlist page. </aside> <p>This year we played a kind of trance that originated in Belgium, Germany, and the Netherlands in the mid 90s.</p> </pre>
<code><h1></code> –h6	A section heading
	<pre> <h1>The Guide To Music On The Playa</h1> <h2>The Main Stage</h2> <p>If you want to play on a stage, you should bring one.</p> <h2>Amplified Music</h2> <p>Amplifiers up to 300W or 90dB are welcome.</p> </pre>

Element	Purpose
	Example
<header>	<pre> <article> <header> <h1>Hard Trance is My Life</h1> <p>By DJ Steve Hill and Technikal</p> </header> <p>The album with the amusing punctuation has red artwork.</p> </article> </pre>
<footer>	<pre> <article> <h1>Hard Trance is My Life</h1> <p>The album with the amusing punctuation has red artwork.</p> <footer> <p>Artists: DJ Steve Hill and Technikal</p> </footer> </article> </pre>
<address>	<pre> <address> To book DJ Steve Hill and Technikal, contact our management. </address> </pre>

§ 4.3.11.1. Article or section?

This section is non-normative.

A [<section>](#) forms part of something else. An [<article>](#) is its own thing. But how does one know which is which? Mostly the real answer is "it depends on author intent".

For example, one could imagine a book with a "Granny Smith" chapter that just said "These juicy, green apples make a great filling for apple pies."; that would be a [<section>](#) because there'd be lots of other chapters on (maybe) other kinds of apples.

On the other hand, one could imagine a tweet or tumblr post or newspaper classified ad that just said "Granny Smith. These juicy, green apples make a great filling for apple pies."; it would then be [<article>](#)s because that was the whole thing.

Comments on an article are not part of the [<article>](#) on which they are commenting, but are related, therefore may be contained in their own nested [<article>](#).

§ 4.4. Grouping content

§ 4.4.1. The p element

Categories:

[Flow content.](#)

[Palpable content.](#)

Contexts in which this element can be used:

Where [flow content](#) is expected.

Content model:

[Phrasing content.](#)

Tag omission in text/html:

A [<p>](#) element's [end tag](#) may be omitted if the [<p>](#) element is immediately followed by an [<address>](#), [<article>](#), [<aside>](#), [<blockquote>](#), [<details>](#), [<div>](#), [<dl>](#), [<fieldset>](#), [<figcaption>](#), [<figure>](#), [<footer>](#), [<form>](#), [<h1>](#), [<h2>](#),

`<h3>`, `<h4>`, `<h5>`, `<h6>`, `<header>`, `<hr>`, `<main>`, `<menu>`, `<nav>`, `<ol>`, `<p>`, `<pre>`, `<section>`, `<table>`, or `<ul>`, element, or if there is no more content in the parent element and the parent element is an [HTML element](#) that is not an `<a>`, `<audio>`, `<del>`, `<ins>`, `<map>`, `<noscript>`, or `<video>` element.

Content attributes:

[Global attributes](#)

Allowed [ARIA role attribute](#) values:

[Any role value.](#)

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLParagraphElement : HTMLElement {};
```

The `<p>` element [represents](#) a [paragraph](#).

NOTE:

While paragraphs are usually represented in visual media by blocks of text that are physically separated from adjacent blocks through blank lines, a style sheet or user agent would be equally justified in presenting paragraph breaks in a different manner, for instance using inline pilcrows (¶).

EXAMPLE 130

The following examples are conforming HTML fragments:

```
<p>The little kitten gently seated itself on a piece of
carpet. Later in his life, this would be referred to as the time the
cat sat on the mat.</p>
```

```
<fieldset>
  <legend>Personal information</legend>
  <p>
    <label>Name: <input name="n"></label>
    <label><input name="anon" type="checkbox"> Hide from other users</label>
  </p>
  <p><label>Address: <textarea name="a"></textarea></label></p>
</fieldset>
```

```
<p>There was once an example from Femley,<br>
Whose markup was of dubious quality.<br>
The validator complained,<br>
So the author was pained,<br>
To move the error from the markup to the rhyming.</p>
```

The `<p>` element should not be used when a more specific element is more appropriate.

EXAMPLE 131

The following example is technically correct:

```
<section>
  <!-- ... -->
  <p>Last modified: 2001-04-23</p>
  <p>Author: fred@example.com</p>
</section>
```

However, it would be better marked-up as:

```
<section>
  <!-- ... -->
  <footer>Last modified: 2001-04-23</footer>
  <address>Author: fred@example.com</address>
</section>
```

Or:

```
<section>
  <!-- ... -->
  <footer>
    <p>Last modified: 2001-04-23</p>
    <address>Author: fred@example.com</address>
  </footer>
</section>
```

List elements (in particular, `ol` and `<ul>` elements) cannot be children of `<p>` elements. When a sentence contains a bulleted list, therefore, one might wonder how it should be marked up.

EXAMPLE 132

For instance, this fantastic sentence has bullets relating to

- wizards,
- faster-than-light travel, and
- telepathy,

and is further discussed below.

The solution is to realize that a paragraph, in HTML terms, is not a logical concept, but a structural one. In the fantastic example above, there are actually five [paragraphs](#) as defined by this specification: one before the list, one for each bullet, and one after the list.

EXAMPLE 133

The markup for the above example could therefore be:

```
<p>For instance, this fantastic sentence has bullets relating to</p>
<ul>
  <li>wizards,
  <li>faster-than-light travel, and
  <li>telepathy,
</ul>
<p>and is further discussed below.</p>
```

Authors wishing to conveniently style such "logical" paragraphs consisting of multiple "structural" paragraphs can use the `<div>` element instead of the `p` element.

EXAMPLE 134

Thus for instance the above example could become the following:

```
<div>For instance, this fantastic sentence has bullets relating to
  <ul>
    <li>wizards,
    <li>faster-than-light travel, and
    <li>telepathy,
  </ul>
  and is further discussed below.</div>
```

This example still has five structural paragraphs, but now the author can style just the `div` instead of having to consider each part of the example separately.

§ 4.4.2. The `hr` element

Categories:

[Flow content.](#)

Contexts in which this element can be used:

Where [flow content](#) is expected.

Content model:

[Nothing.](#)

Tag omission in text/html:

No [end tag](#).

Content attributes:

[Global attributes](#)

Allowed [ARIA role attribute](#) values:

`'separator'` (default - *do not set*) or `'presentation'`.

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

[DOM interface:](#)

```
interface HTMLHRElement : HTMLElement {};
```

The `<hr>` element [represents](#) a [paragraph](#)-level thematic break, e.g., a scene change in a story, or a transition to another topic within a section of a reference book.

EXAMPLE 135

The following fictional extract from a project manual shows two sections that use the `<hr>` element to separate topics within the section.

```
<section>
  <h1>Communication</h1>
  <p>There are various methods of communication. This section
  covers a few of the important ones used by the project.</p>
  <hr>
  <p>Communication stones seem to come in pairs and have mysterious
  properties:</p>
  <ul>
    <li>They can transfer thoughts in two directions once activated
    if used alone.</li>
    <li>If used with another device, they can transfer one's
    consciousness to another body.</li>
    <li>If both stones are used with another device, the
    consciousnesses switch bodies.</li>
  </ul>
  <hr>
  <p>Radios use the electromagnetic spectrum in the meter range and
  longer.</p>
  <hr>
  <p>Signal flares use the electromagnetic spectrum in the
  nanometer range.</p>
</section>
<section>
  <h1>Food</h1>
  <p>All food at the project is rationed:</p>
  <dl>
    <dt>Potatoes</dt>
    <dd>Two per day</dd>
    <dt>Soup</dt>
    <dd>One bowl per day</dd>
  </dl>
  <hr>
  <p>Cooking is done by the chefs on a set rotation.</p>
</section>
```

There is no need for an `<hr>` element between the sections themselves, since the `<section>` elements and the h1 elements imply thematic changes themselves.

EXAMPLE 136

The following extract from *Pandora's Star* by Peter F. Hamilton shows two paragraphs that precede a scene change and the paragraph that follows it. The scene change, represented in the printed book by a gap containing a solitary centered star between the second and third paragraphs, is here represented using the `<hr>` element.

```
<p>Dudley was ninety-two, in his second life, and fast approaching
time for another rejuvenation. Despite his body having the physical
age of a standard fifty-year-old, the prospect of a long degrading
campaign within academia was one he regarded with dread. For a
supposedly advanced civilization, the Intersolar Commonwealth could be
appallingly backward at times, not to mention cruel.</p>
<p><i>Maybe it won't be that bad</i>, he told himself. The lie was
comforting enough to get him through the rest of the night's
shift.</p>
<hr>
<p>The Carlton Alllander drove Dudley home just after dawn. Like the
astronomer, the vehicle was old and worn, but perfectly capable of
doing its job. It had a cheap diesel engine, common enough on a
semi-frontier world like Gralmond, although its drive array was a
thoroughly modern photoneural processor. With its high suspension and
deep-tread tyres it could plough along the dirt track to the
observatory in all weather and seasons, including the metre-deep snow
of Gralmond's winters.</p>
```

NOTE:

The `<hr>` element does not affect the document's [outline](#).

4.4.3. The pre element**Categories:**

[Flow content.](#)

[Palpable content.](#)

Contexts in which this element can be used:

Where [flow content](#) is expected.

Content model:

[Phrasing content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[Any role value.](#)

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLPreElement : HTMLElement {};
```

The `<pre>` element [represents](#) a block of preformatted text, in which structure is represented by typographic conventions rather than by elements.

NOTE:

In [the HTML syntax](#), a leading newline character immediately following the `<pre>` element start tag is stripped.

Some examples of cases where the [<pre>](#) element could be used:

- Including an e-mail, with paragraphs indicated by blank lines, lists indicated by lines prefixed with a bullet, and so on.
- Including fragments of computer code, with structure indicated according to the conventions of that language.
- Displaying ASCII art.

NOTE:

Authors are encouraged to consider how preformatted text will be experienced when the formatting is lost, as will be the case for users of speech synthesizers, braille displays, and the like. For cases like ASCII art, it is likely that an alternative presentation, such as a textual description, would be more universally accessible to the readers of the document.

To represent a block of computer code, the [<pre>](#) element can be used with a [<code>](#) element; to represent a block of computer output the [<pre>](#) element can be used with a [<samp>](#) element. Similarly, the [<kbd>](#) element can be used within a [<pre>](#) element to indicate text that the user is to enter.

NOTE:

This element [has rendering requirements involving the bidirectional algorithm](#).

EXAMPLE 137

In the following snippet, a sample of computer code is presented.

```
<p>This is the <code>Panel</code> constructor:</p>
<pre><code>function Panel(element, canClose, closeHandler) {
  this.element = element;
  this.canClose = canClose;
  this.closeHandler = function () { if (closeHandler) closeHandler() };
}</code></pre>
```

EXAMPLE 138

In the following snippet, [<samp>](#) and [<kbd>](#) elements are mixed in the contents of a [<pre>](#) element to show a session of Zork I.

```
<pre><samp>You are in an open field west of a big white house with a boarded
front door.
There is a small mailbox here.

></samp> <kbd>open mailbox</kbd>

<samp>Opening the mailbox reveals:
A leaflet.

></samp></pre>
```

EXAMPLE 139

The following shows a contemporary poem that uses the `<pre>` element to preserve its unusual formatting, which forms an intrinsic part of the poem itself.

```
<pre>                maxling

it is with a      heart
                heavy

that i admit loss of a feline
                so      loved

a friend lost to the
                unknown

                                (night)

~cdr 11dec07</pre>
```

§ 4.4.4. The `blockquote` element

Categories:

[Flow content.](#)
[Sectioning root.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [flow content](#) is expected.

Content model:

[Flow content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)
 cite - Link to the source of the quotation.

Allowed ARIA role attribute values:

[Any role value.](#)

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)
 Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLQuoteElement : HTMLElement {
  attribute DOMString cite;
};
```

NOTE:

The HTMLQuoteElement interface is also used by the `<q>` element.

The `<blockquote>` element [represents](#) content that is quoted from another source, optionally with a citation which must be within a footer or cite element, and optionally with in-line changes such as annotations and abbreviations.

Content inside a `blockquote` other than citations and in-line changes must be quoted from another source, whose address, if it has one, may be cited in the **cite** attribute.

NOTE:

In cases where a page contains contributions from multiple people, such as comments on a blog post, 'another source' can include text from the same page, written by another person.

If the `cite` attribute is present, it must be a [valid URL potentially surrounded by spaces](#). To obtain the corresponding citation link, the value of the attribute must be [resolved](#) relative to the element. User agents may allow users to follow such citation links, but they are primarily intended for private use (e.g., by server-side scripts collecting statistics about a site's use of quotations), not for readers.

The `cite` IDL attribute must [reflect](#) the element's `cite` content attribute.

The content of a `<blockquote>` may be abbreviated, may have context added or may have annotations. Any such additions or changes to quoted text must be indicated in the text (at the text level). This may mean the use of notational conventions or explicit remarks, such as "emphasis mine".

EXAMPLE 140

For example, in English, abbreviations are traditionally identified using square brackets. Consider a page with the sentence "Fred ate the cracker. He then said he liked apples and fish."; it could be quoted as follows:

```
<blockquote>
  <p>[Fred] then said he liked [...] fish.</p>
</blockquote>
```

Quotation marks may be used to delineate between quoted text and annotations within a `<blockquote>`.

EXAMPLE 141

For example, an in-line note provided by the author:

```
<figure>
<blockquote>
  "That monster custom, who all sense doth eat
  Of habit's devil," <abbr title="et cetera">&c.</abbr> not in Folio

  "What a falling off was there !
  From me, whose love was of that dignity
  That it went hand in hand even with the vow
  I made to her in marriage, and to decline
  Upon a wretch."
</blockquote>
<footer>
  – <cite class="title">Shakespeare manual</cite> by <cite class="author">Frederick Gard
  Fleay</cite>,
  p19 (in Google Books)
</footer>
</figure>
```

NOTE:

In the example above, the citation is contained within the `<footer>` of a `<figure>` element, this groups and associates the information, about the quote, with the quote. The `<figcaption>` element was not used, in this case, as a container for the citation as it is not a caption.

Attribution for the quotation, may be placed inside the `<blockquote>` element, but must be within a `<cite>` element for in-text attributions or within a `<footer>` element.

EXAMPLE 142

For example, here the attribution is given in a footer after the quoted text, to clearly relate the quote to its attribution:

```
<blockquote>
  <p>I contend that we are both atheists. I just believe in one fewer
  god than you do. When you understand why you dismiss all the other
  possible gods, you will understand why I dismiss yours.</p>
  <footer>- <cite>Stephen Roberts</cite></footer>
</blockquote>
```

EXAMPLE 143

Here the attribution is given in a `<cite>` element on the last line of the quoted text. Note that a link to the author is also included.

```
<blockquote>
  The people recognize themselves in their commodities; they find their
  soul in their automobile, hi-fi set, split-level home, kitchen equipment.
  - <cite><a href="https://en.wikipedia.org/wiki/Herbert_Marcuse">Herbert Marcuse</a></cite>
</blockquote>
```

NOTE:

There is no formal method for indicating the markup in a `blockquote` is from a quoted source. It is suggested that if the footer or `<cite>` elements are included and these elements are also being used within a `blockquote` to identify citations, the elements from the quoted source could be annotated with metadata to identify their origin, for example by using the `class` attribute (a defined extensibility mechanism).

EXAMPLE 144

In this example the source of a quote includes a `<cite>` element, which is annotated using the `class` attribute:

```
<blockquote>
  <p>My favorite book is <cite class="from-source">At Swim-Two-Birds</cite></p>
  <footer>- <cite>Mike[tm]Smith</cite></footer>
</blockquote>
```

The other examples below show other ways of showing attribution.

EXAMPLE 145

Here a `<blockquote>` element is used in conjunction with a `<figure>` element and its `figcaption`:

```
<figure>
  <blockquote>
    <p>The truth may be puzzling. It may take some work to grapple with.
    It may be counterintuitive. It may contradict deeply held
    prejudices. It may not be consonant with what we desperately want to
    be true. But our preferences do not determine what's true. We have a
    method, and that method helps us to reach not absolute truth, only
    asymptotic approaches to the truth – never there, just closer
    and closer, always finding vast new oceans of undiscovered
    possibilities. Cleverly designed experiments are the key.</p>
  </blockquote>
  <figcaption><cite>Carl Sagan</cite>, in "<cite>Wonder and Skepticism</cite>", from
  the <cite>Skeptical Inquirer</cite> Volume 19, Issue 1 (January-February
  1995)</figcaption>
</figure>
```

EXAMPLE 146

This next example shows the use of `cite` alongside `blockquote`:

```
<p>His next piece was the aptly named <cite>Sonnet 130</cite>:</p>
<blockquote cite="https://quotes.example.org/s/sonnet130.html">
  <p>My mistress' eyes are nothing like the sun,<br>
  Coral is far more red, than her lips red,<br>
  ...
```

EXAMPLE 147

This example shows how a forum post could use `blockquote` to show what post a user is replying to. The `<article>` element is used for each post, to mark up the threading.

```
<article>
  <h1><a href="https://bacon.example.com/?blog=109431">Bacon on a crowbar</a></h1>
  <article>
    <header><strong>t3yw</strong> 12 points 1 hour ago</header>
    <p>I bet a narwhal would love that.</p>
    <footer><a href="?pid=29578">permalink</a></footer>
  </article>
  <article>
    <header><strong>greg</strong> 8 points 1 hour ago</header>
    <blockquote><p>I bet a narwhal would love that.</p></blockquote>
    <p>Dude narwhals don't eat bacon.</p>
    <footer><a href="?pid=29579">permalink</a></footer>
  </article>
  <article>
    <header><strong>t3yw</strong> 15 points 1 hour ago</header>
    <blockquote>
      <blockquote><p>I bet a narwhal would love that.</p></blockquote>
      <p>Dude narwhals don't eat bacon.</p>
    </blockquote>
    <p>Next thing you'll be saying they don't get capes and wizard hats either!</p>
    <footer><a href="?pid=29580">permalink</a></footer>
  </article>
  <article>
    <header><strong>boing</strong> -5 points 1 hour ago</header>
    <p>narwhals are worse than ceiling cat</p>
    <footer><a href="?pid=29581">permalink</a></footer>
  </article>
</article>
<article>
  <header><strong>fred</strong> 1 points 23 minutes ago</header>
  <blockquote><p>I bet a narwhal would love that.</p></blockquote>
  <p>I bet they'd love to peel a banana too.</p>
  <footer><a href="?pid=29582">permalink</a></footer>
</article>
</article>
```

EXAMPLE 148

This example shows the use of a blockquote for short snippets, demonstrating that one does not have to use `<p>` elements inside `<blockquote>` elements:

```
<p>He began his list of "lessons" with the following:</p>
<blockquote>One should never assume that his side of
the issue will be recognized, let alone that it will
be conceded to have merits.</blockquote>
<p>He continued with a number of similar points, ending with:</p>
<blockquote>Finally, one should be prepared for the threat
of breakdown in negotiations at any given moment and not
be cowed by the possibility.</blockquote>
<p>We shall now discuss these points...
```

NOTE:

Examples of how to represent a conversation are shown in a later section; it is not appropriate to use the `cite` and `<blockquote>` elements for this purpose.

4.4.5. The `ol` element**Categories:**

[Flow content.](#)

If the element's children include at least one `<li>` element: [Palpable content.](#)

Contexts in which this element can be used:

Where [flow content](#) is expected.

Content model:

Zero or more `li` and [script-supporting elements](#).

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

`reversed` - Number the list backwards.

`start` - [Ordinal value](#) of the first item

`type` - Kind of list marker.

Allowed ARIA role attribute values:

`'list'` role (default - *do not set*), `'directory'`, `'group'`, `'listbox'`, `'menu'`, `'menubar'`, `'presentation'`, `'radiogroup'`, `'tablist'`, `'toolbar'` or `'tree'`.

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLListElement : HTMLElement {
  attribute boolean reversed;
  attribute long start;
  attribute DOMString type;
};
```

The `<ol>` element [represents](#) a list of items, where the items have been intentionally ordered, such that changing the order would change the meaning of the document.

The items of the list are the `<li>` element child nodes of the `<ol>` element, in [tree order](#).

The **reversed** attribute is a [boolean attribute](#). If present, it indicates that the list is a descending list (... , 3, 2, 1). If the attribute is omitted, the list is an ascending list (1, 2, 3, ...).

The **start** attribute, if present, must be a [valid integer](#) giving the [ordinal value](#) of the first list item.

If the **start** attribute is present, user agents must [parse it as an integer](#), in order to determine the attribute's value. The default value, used if the attribute is missing or if the value cannot be converted to a number according to the referenced algorithm, is 1 if the element has no **reversed** attribute, and is the number of child **li** elements otherwise.

The first item in the list has the [ordinal value](#) given by the `<ol>` element's **start** attribute, unless that `<li>` element has a **value** attribute with a value that can be successfully parsed, in which case it has the [ordinal value](#) given by that **value** attribute.

Each subsequent item in the list has the [ordinal value](#) given by its **value** attribute, if it has one, or, if it doesn't, the [ordinal value](#) of the previous item, plus one if the **reversed** is absent, or minus one if it is present.

The **type** attribute can be used to specify the kind of marker to use in the list, in the cases where that matters (e.g., because items are to be referenced by their number/letter). The attribute, if specified, must have a value that is a [case-sensitive](#) match for one of the characters given in the first cell of one of the rows of the following table. The **type** attribute represents the state given in the cell in the second column of the row whose first cell matches the attribute's value; if none of the cells match, or if the attribute is omitted, then the attribute represents the [decimal](#) state.

Keyword	State	Description	Examples for values 1-3 and 3999-4001						
1 (U+0031)	decimal	Decimal numbers	1.	2.	3.	...	3999.	4000.	4001.
a (U+0061)	lower-alpha	Lowercase latin alphabet	a.	b.	c.	...	ewu.	ewv.	eww.
A (U+0041)	upper-alpha	Uppercase latin alphabet	A.	B.	C.	...	EWU.	EWV.	EWV.
i (U+0069)	lower-roman	Lowercase roman numerals	i.	ii.	iii.	...	mmcmxcix.	iv.	ivi.
I (U+0049)	upper-roman	Uppercase roman numerals	I.	II.	III.	...	MMCMXCIX.	IV.	IVI.

User agents should render the items of the list in a manner consistent with the state of the **type** attribute of the `<ol>` element. Numbers less than or equal to zero should always use the decimal system regardless of the **type** attribute.

NOTE:

For CSS user agents, a mapping for this attribute to the [‘list-style-type’](#) CSS property is given in the [§10 Rendering](#) section (the mapping is straightforward: the states above have the same names as their corresponding CSS values).

NOTE:

It is possible to redefine the default CSS list styles used to implement this attribute in CSS user agents; doing so will affect how list items are rendered.

The **reversed**, **start**, and **type** IDL attributes must [reflect](#) the respective content attributes of the same name. The **start** IDL attribute has the same default as its content attribute.

EXAMPLE 149

The following markup shows a list where the order matters, and where the `ol` element is therefore appropriate. Compare this list to the equivalent list in the `ul` section to see an example of the same items using the `<ul>` element.

```
<p>I have lived in the following countries (given in the order of when
I first lived there):</p>
<ol>
  <li>Switzerland
  <li>United Kingdom
  <li>United States
  <li>Norway
</ol>
```

Note how changing the order of the list changes the meaning of the document. In the following example, changing the relative order of the first two items has changed the birthplace of the author:

```
<p>I have lived in the following countries (given in the order of when
I first lived there):</p>
<ol>
  <li>United Kingdom
  <li>Switzerland
  <li>United States
  <li>Norway
</ol>
```

4.4.6. The `ul` element**Categories:**

[Flow content.](#)

If the element's children include at least one `<li>` element: [Palpable content.](#)

Contexts in which this element can be used:

Where [flow content](#) is expected.

Content model:

Zero or more `li` and [script-supporting elements](#).

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

'list' role (default - *do not set*), 'directory', 'group', 'listbox', 'menu', 'menubar', 'presentation', 'radiogroup', 'tablist', 'toolbar' or 'tree'.

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLULListElement : HTMLElement {};
```

The `<ul>` element [represents](#) a list of items, where the order of the items is not important — that is, where changing the order would not materially change the meaning of the document.

The items of the list are the `<li>` element child nodes of the `<ul>` element.

EXAMPLE 150

The following markup shows a list where the order does not matter, and where the `ul` element is therefore appropriate. Compare this list to the equivalent list in the `ol` section to see an example of the same items using the `<ol>` element.

```
<p>I have lived in the following countries:</p>
<ul>
  <li>Norway
  <li>Switzerland
  <li>United Kingdom
  <li>United States
</ul>
```

Note that changing the order of the list does not change the meaning of the document. The items in the snippet above are given in alphabetical order, but in the snippet below they are given in order of the size of their current account balance in 2007, without changing the meaning of the document whatsoever:

```
<p>I have lived in the following countries:</p>
<ul>
  <li>Switzerland
  <li>Norway
  <li>United Kingdom
  <li>United States
</ul>
```

§ 4.4.7. The `li` element**Categories:**

None.

Contexts in which this element can be used:

Inside `<ol>` elements.

Inside `<ul>` elements.

Content model:

Flow content.

Tag omission in text/html:

An `<li>` element's **end tag** may be omitted if the `<li>` element is immediately followed by another `<li>` element or if there is no more content in the parent element.

Content attributes:

Global attributes

If the element is not a child of an `<ul>` or `<menu>` element: `value`

Allowed ARIA role attribute values:

`'listitem'` role (default - *do not set*), `'menuitem'`, `'menuitemcheckbox'`, `'menuitemradio'`, `'option'`, `'presentation'`, `'radio'`, `'separator'`, `'tab'` or `'treeitem'`.

Allowed ARIA state and property attributes:

Global aria-* attributes

Any aria-* attributes **applicable to the allowed roles**.

DOM interface:

```
interface HTMLLIElement : HTMLElement {
  attribute long value;
};
```

The `<li>` element **represents** a list item. If its parent element is an `<ol>`, `<ul>`, or `<menu>` element, then the element is an item of the parent element's list, as defined for those elements. Otherwise, the list item has no defined list-related relationship to

any other `<li>` element.

If the parent element is an `<ol>` element, then the `<li>` element has an **ordinal value**.

The **value** attribute, if present, must be a [valid integer](#) giving the [ordinal value](#) of the list item.

If the **value** attribute is present, user agents must [parse it as an integer](#), in order to determine the attribute's value. If the attribute's value cannot be converted to a number, the attribute must be treated as if it was absent. The attribute has no default value.

The **value** attribute is processed relative to the element's parent `ol` element (q.v.), if there is one. If there is not, the attribute has no effect.

The **value** IDL attribute must [reflect](#) the value of the **value** content attribute.

EXAMPLE 151

The following example, the top ten movies are listed (in reverse order). Note the way the list is given a title by using a `<figure>` element and its `<figcaption>` element.

```
<figure>
  <figcaption>The top 10 movies of all time</figcaption>
  <ol>
    <li value="10"><cite>Josie and the Pussycats</cite>, 2001</li>
    <li value="9"><cite lang="sh">Црна мачка, бели мачор</cite>, 1998</li>
    <li value="8"><cite>A Bug's Life</cite>, 1998</li>
    <li value="7"><cite>Toy Story</cite>, 1995</li>
    <li value="6"><cite>Monsters, Inc</cite>, 2001</li>
    <li value="5"><cite>Cars</cite>, 2006</li>
    <li value="4"><cite>Toy Story 2</cite>, 1999</li>
    <li value="3"><cite>Finding Nemo</cite>, 2003</li>
    <li value="2"><cite>The Incredibles</cite>, 2004</li>
    <li value="1"><cite>Ratatouille</cite>, 2007</li>
  </ol>
</figure>
```

The markup could also be written as follows, using the `reversed` attribute on the `<ol>` element:

```
<figure>
  <figcaption>The top 10 movies of all time</figcaption>
  <ol reversed>
    <li><cite>Josie and the Pussycats</cite>, 2001</li>
    <li><cite lang="sh">Црна мачка, бели мачор</cite>, 1998</li>
    <li><cite>A Bug's Life</cite>, 1998</li>
    <li><cite>Toy Story</cite>, 1995</li>
    <li><cite>Monsters, Inc</cite>, 2001</li>
    <li><cite>Cars</cite>, 2006</li>
    <li><cite>Toy Story 2</cite>, 1999</li>
    <li><cite>Finding Nemo</cite>, 2003</li>
    <li><cite>The Incredibles</cite>, 2004</li>
    <li><cite>Ratatouille</cite>, 2007</li>
  </ol>
</figure>
```

NOTE:

While it is conforming to include heading elements (e.g., `h1`) inside `li` elements, it likely does not convey the semantics that the author intended. A heading starts a new section, so a heading in a list implicitly splits the list into spanning multiple sections.

Categories:

[Flow content](#).

If the element's children include at least one name-value group: [Palpable content](#).

Contexts in which this element can be used:

Where [flow content](#) is expected.

Content model:

Zero or more groups each consisting of one or more [<dt>](#) elements followed by one or more [<dd>](#) elements, optionally intermixed with [script-supporting elements](#).

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

'list' role (default - *do not set*), ['directory'](#), ['group'](#), ['listbox'](#), ['menu'](#), ['menubar'](#), ['presentation'](#), ['radiogroup'](#), ['tablist'](#), ['toolbar'](#) or ['tree'](#).

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLDListElement : HTMLElement {};
```

The [<dl>](#) element [represents](#) a description list of zero or more term-description groups. Each term-description group consists of one or more terms (represented by [<dt>](#) elements), and one or more descriptions (represented by [<dd>](#) elements).

Term-description groups may be names and definitions, questions and answers, categories and topics, or any other groups of term-description pairs.

EXAMPLE 152

In this example a [<dl>](#) is used to represent a simple list of names and descriptions:

```
<dl>
  <dt>Blanco tequila</dt>
  <dd>The purest form of the blue agave spirit...</dd>
  <dt>Reposado tequila</dt>
  <dd>Typically aged in wooden barrels for between two and eleven months...</dd>
</dl>
```

Each term within a term-description group must be represented by a single [<dt>](#) element. The descriptions within a term-description group are alternatives. Each description must be represented by a single [<dd>](#) element.

EXAMPLE 153

In this example a `<dl>` element represents a set of terms, each of which has multiple descriptions:

```
<p>Information about the rock band Queen:</p>
<dl>
  <dt>Members</dt>
  <dd>Brian May</dd>
  <dd>Freddie Mercury</dd>
  <dd>John Deacon</dd>
  <dd>Roger Taylor</dd>
  <dt>Record labels</dt>
  <dd>EMI</dd>
  <dd>Parlophone</dd>
  <dd>Capitol</dd>
  <dd>Hollywood</dd>
  <dd>Island</dd>
</dl>
```

The order of term-description groups within a `<dl>` element, and the order of terms and descriptions within each group, may be significant.

EXAMPLE 154

In this example a `<dl>` is used to show a set of instructions, where the order of the instructions is important:

```
<p>Determine the victory points as follows (use the first matching case):</p>
<dl>
  <dt> If you have exactly five gold coins </dt>
  <dd> You get five victory points </dd>
  <dt> If you have one or more gold coins, and you have one or more silver coins </dt>
  <dd> You get two victory points </dd>
  <dt> If you have one or more silver coins </dt>
  <dd> You get one victory point </dd>
  <dt> Otherwise </dt>
  <dd> You get no victory points </dd>
</dl>
```

If a `<dl>` element contains no `<dt>` or `<dd>` child elements, it contains no term-description groups.

If a `<dl>` element has one or more non-`whitespace` text node children, or has children that are neither `<dt>` or `<dd>` elements, then all such text nodes and elements as well as their descendants (including any `<dt>` and `<dd>` elements) do not form part of any term-description group within the `<dl>`.

If a `<dl>` element has one or more `<dt>` element children, but no `<dd>` element children, then it consists of one group with terms but no descriptions.

If a `<dl>` element has one or more `<dd>` element children, but no `<dt>` element children, it consists of one group with descriptions but no terms.

If a `<dd>` element is the first child of a `<dl>` element (excepting a [script-supporting element](#)), the first group has no associated term.

If a `<dt>` element is the last child of a `<dl>` element (excepting a [script-supporting element](#)), the last group has no associated descriptions.

NOTE:

Note: when a `<dl>` element does not match its content model, it is often because a `<dd>` element has been used instead of a `<dt>` element, or vice versa.

§ 4.4.9. The dt element

Categories:

None.

Contexts in which this element can be used:

Before `dd` or `<dt>` elements inside `<dl>` elements.

Content model:

Flow content, but with no `<header>`, `<footer>`, `sectioning content`, or `heading content` descendants.

Tag omission in text/html:

A `<dt>` element's [end tag](#) may be omitted if the `<dt>` element is immediately followed by another `<dt>` element or a `<dd>` element.

Content attributes:

[Global attributes](#)

Allowed [ARIA role attribute](#) values:

None.

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes None

DOM interface:

Uses `HTMLElement`.

The `<dt>` element [represents](#) a term, part of a term-description group in a description list (`<dl>` element).

EXAMPLE 155

In this example the `<dt>` elements represent questions and the `<dd>` elements the answers:

```
<dl>
  <dt>What is my favorite drink?</dt>
  <dd>Tea</dd>
  <dt>What is my favorite food?</dt>
  <dd>Sushi</dd>
  <dt>What is my favourite film?</dt>
  <dd>What a Wonderful Life</dd>
</dl>
```

NOTE:

When used within a `<dl>` element, the `<dt>` element does not necessarily represent the definition for a term. The `<dfn>` element should be used to represent a definition.

EXAMPLE 156

In this example the `<dfn>` element indicates that the `<dt>` element contains a defined term, the definition for which is represented by the `<dd>` element:

```
<dl>
  <dt lang="en-us"><dfn>Color</dfn></dt>
  <dt lang="en-gb"><dfn>Colour</dfn></dt>
  <dd>A sensation which (in humans) derives from the ability of the fine structure of the eye
  to distinguish three differently filtered analyses of a view.</dd>
</dl>
```

§ 4.4.10. The dd element

Categories:

None.

Contexts in which this element can be used:

After `dt` or `<dd>` elements inside `<dl>` elements.

Content model:

Flow content.

Tag omission in text/html:

A `<dd>` element's [end tag](#) may be omitted if the `<dd>` element is immediately followed by another `<dd>` element or a `<dt>` element, or if there is no more content in the parent element.

Content attributes:

[Global attributes](#)

Allowed [ARIA role attribute](#) values:

None

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

Uses `HTMLElement`.

The `<dd>` element [represents](#) a description, part of a term-description group in a description list (`<dl>` element).

EXAMPLE 157

In this example the `<dd>` elements represent the keys that invoke the keycodes indicated in the `<dt>` elements:

```
<dl>
  <dt>37</dt>
  <dd>Left</dd>
  <dt>38</dt>
  <dd>Right</dd>
  <dt>39</dt>
  <dd>Up</dd>
  <dt>40</dt>
  <dd>Down</dd>
</dl>
```

§ 4.4.11. The `figure` element**Categories:**

[Flow content](#).

[Sectioning root](#).

[Palpable content](#).

Contexts in which this element can be used:

Where [flow content](#) is expected.

Content model:

[Flow content](#) optionally including a `<figcaption>` child element.

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed [ARIA role attribute](#) values:

[Any role value.](#)

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

Uses `HTMLElement`.

The `<figure>` element [represents](#) some [flow content](#), optionally with a caption, that is self-contained (like a complete sentence) and is typically referenced as a single unit from the main flow of the document.

NOTE:

Self-contained in this context does not necessarily mean independent. For example, each sentence in a paragraph is self-contained; an image that is part of a sentence would be inappropriate for `<figure>`, but an entire sentence made of images would be fitting.

The element can thus be used to annotate illustrations, diagrams, photos, code listings, etc.

When a figure is referred to from the main content of the document by identifying it by its caption (e.g., by figure number), it enables such content to be easily moved away from that primary content, e.g., to the side of the page, to dedicated pages, or to an appendix, without affecting the flow of the document.

If a `<figure>` element is referenced by its relative position, e.g., "in the photograph above" or "as the next figure shows", then moving the figure would disrupt the page's meaning. Authors are encouraged to consider using labels to refer to figures, rather than using such relative references, so that the page can easily be restyled without affecting the page's meaning.

The `<figcaption>` descendant of `<figure>`, if any, represents the caption of the `<figure>` element's contents. If there is no child `<figcaption>` element, then there is no caption.

A `<figure>` element's contents are part of the surrounding flow. If the purpose of the page is to display the figure, for example a photograph on an image sharing site, the `figure` and `<figcaption>` elements can be used to explicitly provide a caption for that figure. For content that is only tangentially related, or that serves a separate purpose than the surrounding flow, the `<aside>` element should be used (and can itself wrap a `<figure>`). For example, a pull quote that repeats content from an `<article>` would be more appropriate in an `<aside>` than in a `<figure>`, because it isn't part of the content, it's a repetition of the content for the purposes of enticing readers or highlighting key topics.

EXAMPLE 158

This example shows the `<figure>` element to mark up a code listing.

```
<p>In listing 4 we see the primary core interface
API declaration.</p>
<figure id="l4">
  <figcaption>Listing 4. The primary core interface API declaration.</figcaption>
  <pre><code>interface PrimaryCore {
    boolean verifyDataLine();
    void sendData(in sequence<byte> data);
    void initSelfDestruct();
  }</code></pre>
</figure>
<p>The API is designed to use UTF-8.</p>
```


EXAMPLE 159

Here we see a `<figure>` element to mark up a photo that is the main content of the page (as in a gallery).

```
<!DOCTYPE HTML>
<title>Bubbles at work – My Gallery™</title>
<figure>
  
  <figcaption>Bubbles at work</figcaption>
</figure>
<nav><a href="19414.html">Prev</a> – <a href="19416.html">Next</a></nav>
```

EXAMPLE 160

In this example, we see an image that is *not* a figure, as well as an image and a video that are. The first image is literally part of the example's second sentence, so it's not a self-contained unit, and thus `figure` would be inappropriate.

```
<h2>Malinko's comics</h2>

<p>This case centered on some sort of "intellectual property"
infringement related to a comic (see Exhibit A). The suit started
after a trailer ending with these words:

<blockquote>
  
</blockquote>

<p>...was aired. A lawyer, armed with a Bigger Notebook, launched a
preemptive strike using snowballs. A complete copy of the trailer is
included with Exhibit B.

<figure>
  
  <figcaption>Exhibit A. The alleged <cite>rough copy</cite> comic.</figcaption>
</figure>

<figure>
  <video src="ex-b.mov"></video>
  <figcaption>Exhibit B. The <cite>Rough Copy</cite> trailer.</figcaption>
</figure>

<p>The case was resolved out of court.
```

EXAMPLE 161

Here, a part of a poem is marked up using `<figure>`.

```
<figure>
  <p>'Twas brillig, and the slithy toves<br>
    Did gyre and gimble in the wabe;<br>
    All mimsy were the borogoves,<br>
    And the mome raths outgrabe.</p>
  <figcaption><cite>Jabberwocky</cite> (first verse). Lewis Carroll, 1832-98</figcaption>
</figure>
```

EXAMPLE 162

In this example, which could be part of a much larger work discussing a castle, nested `<figure>` elements are used to provide both a group caption and individual captions for each figure in the group:

```
<figure>
  <figcaption>The castle through the ages: 1423, 1858, and 1999 respectively.</figcaption>
  <figure>
    <figcaption>Etching. Anonymous, ca. 1423.</figcaption>
    
  </figure>
  <figure>
    <figcaption>Oil-based paint on canvas. Maria Towle, 1858.</figcaption>
    
  </figure>
  <figure>
    <figcaption>Film photograph. Peter Jankle, 1999.</figcaption>
    
  </figure>
</figure>
```

EXAMPLE 163

The previous example could also be more succinctly written as follows (using `title` attributes in place of the nested `figure/figcaption` pairs):

```
<figure>
  
  
  
  <figcaption>The castle through the ages: 1423, 1858, and 1999 respectively.</figcaption>
</figure>
```

EXAMPLE 164

The figure is sometimes referenced only implicitly from the content:

```
<article>
  <h1>Fiscal negotiations stumble in Congress as deadline nears</h1>
  <figure>
    
    <figcaption>Barack Obama and Harry Reid. White House press photograph.</figcaption>
  </figure>
  <p>Negotiations in Congress to end the fiscal impasse sputtered on Tuesday, leaving both
chambers
  grasping for a way to reopen the government and raise the country's borrowing authority with
a
  Thursday deadline drawing near.</p>
  ...
</article>
```

§ 4.4.12. The `figcaption` element**Categories:**

None.

Contexts in which this element can be used:

As a descendant of a [<figure>](#) element.

Content model:

[Flow content](#).

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[Any role value](#).

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

Uses `HTMLElement`.

The [<figcaption>](#) element [represents](#) a caption or legend for the rest of the contents of the [<figcaption>](#) element's parent [<figure>](#) element, if any.

4.4.13. The main element

Categories:

[Flow content](#).

[Palpable content](#).

Contexts in which this element can be used:

Where [flow content](#) is expected, but with no [<article>](#), [<aside>](#), [<footer>](#), [<header>](#) or [<nav>](#) element ancestors.

Content model:

[Flow content](#).

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[‘main’](#) role (default - *do not set*) or [‘presentation’](#).

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

Uses `HTMLElement`

The [<main>](#) element [represents](#) the main content of the [<body>](#) of a document or application.

NOTE:

The [<main>](#) element is not [sectioning content](#) and has no effect on the document [outline](#)

The main content area of a document includes content that is unique to that document and excludes content that is repeated across a set of documents such as site navigation links, copyright information, site logos and banners and search forms (unless the document or application's main function is that of a search form).

Authors must not include the [<main>](#) element as a descendant of an [<article>](#), [<aside>](#), [<footer>](#), [<header>](#) or [<nav>](#) element.

NOTE:

The `<main>` element is not suitable for use to identify the main content areas of sub sections of a document or application. The simplest solution is to not mark up the main content of a sub section at all, and just leave it as implicit, but an author could use a [§4.4 Grouping content](#) or [sectioning content](#) element as appropriate.

In the following example, we see 2 articles about skateboards (the main topic of a Web page) the main topic content is identified by the use of the `<main>` element.

EXAMPLE 165

```
<!-- other content -->

<main>

  <h1>Skateboards</h1>
  <p>The skateboard is the way cool kids get around</p>

  <article>
    <h2>Longboards</h2>
    <p>Longboards are a type of skateboard with a longer
    wheelbase and larger, softer wheels.</p>
    <p>... </p>
    <p>... </p>
  </article>

  <article>
    <h2>Electric Skateboards</h2>
    <p>These no longer require the propelling of the skateboard
    by means of the feet; rather an electric motor propels the board,
    fed by an electric battery.</p>
    <p>... </p>
    <p>... </p>
  </article>

</main>

<!-- other content -->
```

Here is a graduation programme the main content section is defined by the use of the `<main>` element. Note in this example the `<main>` element contains a `<nav>` element consisting of links to sub sections of the main content.

EXAMPLE 166

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Graduation Ceremony Summer 2022</title>
  </head>
  <body>

    <header>The Lawson Academy:
    <nav>
      <ul>
        <li><a href="courses.html">Courses</a></li>
        <li><a href="fees.html">Fees</a></li>
        <li><a>Graduation</a></li>
      </ul>
    </nav>
  </header>

  <main>

    <h1>Graduation</h1>

    <nav>
      <ul>
        <li><a href="#ceremony">Ceremony</a></li>
        <li><a href="#graduates">Graduates</a></li>
        <li><a href="#awards">Awards</a></li>
      </ul>
    </nav>

    <h2 id="ceremony">Ceremony</h2>
    <p>Opening Procession</p>
    <p>Speech by Valedictorian</p>
    <p>Speech by Class President</p>
    <p>Presentation of Diplomas</p>
    <p>Closing Speech by Headmaster</p>

    <h2 id="graduates">Graduates</h2>
    <ul>
      <li>Eileen Williams</li>
      <li>Andy Maseyk</li>
      <li>Blanca Sainz Garcia</li>
      <li>Clara Faulkner</li>
      <li>Gez Lemon</li>
      <li>Eloisa Faulkner</li>
    </ul>

    <h2 id="awards">Awards</h2>
    <ul>
      <li>Clara Faulkner</li>
      <li>Eloisa Faulkner</li>
      <li>Blanca Sainz Garcia</li>
    </ul>

  </main>

  <footer> Copyright 2012 B.lawson</footer>

</body>
</html>

```

In the next example, both the `<header>` and the `<footer>` are outside the `<main>` element because they are generic to the web-

site and not specific to `<main>`'s content.

EXAMPLE 167

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Great Dogs for Families</title>
  </head>
  <body>

    <header>
      <h1>The Border Terrier</h1>
      <nav>
        <ul>
          <li><a href="index.html">Home</a></li>
          <li><a href="about.html">About</a></li>
          <li><a href="health.html">Health</a></li>
        </ul>
      </nav>
    </header>
    <main>
      <h2>Welcome!</h2>
      <p>This site is all about the Border Terrier, the best breed of dog
        that there is!</p>
    </main>
    <footer>
      <small>Copyright © <time datetime="2013">2013</time> by I. Devlin</small>
    </footer>

  </body>
</html>
```

Here, the same generic `<header>` and `<footer>` elements remain outside `<main>`, but there is an additional `<header>` element within the `<main>` element as its content is relevant to the content within `<main>` because it contains a relevant heading and in-page navigation. The in-page navigation is repeated within a `<footer>` which is again within the `<main>` element.

EXAMPLE 168

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Great Dogs for Families</title>
  </head>
  <body>

    <header>
      <h1>The Border Terrier</h1>
      <nav>
        <ul>
          <li><a href="index.html">Home</a></li>
          <li><a href="about.html">About</a></li>
          <li><a href="health.html">Health</a></li>
        </ul>
      </nav>
    </header>
    <main>
      <section>
        <header>
          <h2>About</h2>
          <nav>
            <ul>
              <li><a href="#basic">Basic</a></li>
              <li><a href="#app">Appearance</a></li>
              <li><a href="#temp">Temperament</a></li>
            </ul>
          </nav>
        </header>
        <section id="basic">
          <h3>Basic Information</h3>
          <p>The Border Terrier is a small, rough-coated breed of
            dog of the terrier group, originally bred as fox and vermin
            hunters. [...]</p>
        </section>
        <section id="app">
          <h3>Appearance</h3>
          <p>Identifiable by their otter-shaped heads, Border Terriers
            have a broad skull and short (although many be fairly long),
            strong muzzle with a scissors bite. [...]</p>
        </section>
        <section id="temp">
          <h3>Temperament</h3>
          <p>Though sometimes stubborn and strong willed, border terriers
            are, on the whole very even tempered, and are friendly and rarely
            aggressive. [...]</p>
        </section>
        <footer>
          <a href="#basic">Basic</a> -
          <a href="#app">Appearance</a> -
          <a href="#temp">Temperament</a>
        </footer>
      </section>
    </main>
    <footer>
      <small>Copyright © <time datetime="2013">2013</time> by I. Devlin</small>
    </footer>

  </body>
</html>

```

This example is largely the same as the previous one except that it includes an `<aside>`. The content of the `<aside>` is considered to be relevant to the content within the `<main>` element, which is all about the Border Terrier, so the `<aside>` is placed within the `<main>` element.

EXAMPLE 169

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Great Dogs for Families</title>
  </head>
  <body>

    <header>
      <h1>The Border Terrier</h1>
      <nav>
        <ul>
          <li><a href="index.html">Home</a></li>
          <li><a href="about.html">About</a></li>
          <li><a href="health.html">Health</a></li>
        </ul>
      </nav>
    </header>
    <main>
      <section>
        <header>
          <h2>About</h2>
          <nav>
            <ul>
              <li><a href="#basic">Basic</a></li>
              <li><a href="#app">Appearance</a></li>
              <li><a href="#temp">Temperament</a></li>
            </ul>
          </nav>
        </header>
        <section id="basic">
          <h3>Basic Information</h3>
          <p>The Border Terrier is a small, rough-coated breed of dog of the terrier group, originally bred as fox and vermin hunters. [...]</p>
        </section>
        <section id="app">
          <h3>Appearance</h3>
          <p>Identifiable by their otter-shaped heads, Border Terriers have a broad skull and short (although many be fairly long), strong muzzle with a scissors bite. [...]</p>
        </section>
        <section id="temp">
          <h3>Temperament</h3>
          <p>Though sometimes stubborn and strong willed, border terriers are, on the whole very even tempered, and are friendly and rarely aggressive. [...] </p>
        </section>
        <aside>
          <h3>History</h3>
          <p>The Border Terrier originates in, and takes its name from the Scottish borders. [...] </p>
        </aside>
        <footer>
          <a href="#basic">Basic</a> -
          <a href="#app">Appearance</a> -
          <a href="#temp">Temperament</a>
        </footer>
      </section>
    </main>
    <footer>
      <small>Copyright © <time datetime="2013">2013</time> by I. Devlin</small>
    </footer>
  </body>
</html>
```

```

    </body>
  </html>

```

In the following example, two `<aside>` elements containing adverts have been placed outside the `<main>` element as their content is not specific to the content within `<main>`. These `<aside>`s could be on any page, as they are as generic as the `<header>` and `<footer>` shown.

EXAMPLE 170

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Great Dogs for Families</title>
  </head>
  <body>

    <header>
      <h1>The Border Terrier</h1>
      <nav>
        <ul>
          <li><a href="index.html">Home</a></li>
          <li><a href="about.html">About</a></li>
          <li><a href="health.html">Health</a></li>
        </ul>
      </nav>
    </header>
    <main>
      <h2>Welcome!</h2>
      <p>This site is all about the Border Terrier, the best breed of dog
that there is!</p>
    </main>
    <aside class="advert">
      <h2>Border Farm Breeders</h2>
      <p>We are a certified breeder of Border Terriers, contact us at...</p>
    </aside>
    <aside class="advert">
      <h2>Grumpy's Pet Shop</h2>
      <p>Get all your pet's needs at our shop!</p>
    </aside>

    <footer>
      <small>Copyright © <time datetime="2013">2013</time> by I. Devlin</small>
    </footer>

  </body>
</html>

```

4.4.14. The `div` element

Categories:

[Flow content.](#)

[Palpable content.](#)

Contexts in which this element can be used:

Where [flow content](#) is expected.

Content model:

[Flow content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:[Global attributes](#)**Allowed [ARIA role attribute](#) values:**[Any role value.](#)**Allowed [ARIA state and property attributes](#):**[Global aria-* attributes](#)Any aria-* attributes [applicable to the allowed roles](#).**DOM interface:**

```
interface HTMLDivElement : HTMLElement {};
```

The `<div>` element has no special meaning at all. It [represents](#) its children. It can be used with the `class`, [lang](#), and `title` attributes to mark up semantics common to a group of consecutive elements.

NOTE:

Authors are strongly encouraged to view the `<div>` element as an element of last resort, for when no other element is suitable. Use of more appropriate elements instead of the `<div>` element leads to better accessibility for readers and easier maintainability for authors.

EXAMPLE 171

For example, a blog post would be marked up using `<article>`, a chapter using `<section>`, a page's navigation aids using `<nav>`, and a group of form controls using `<fieldset>`.

On the other hand, `<div>` elements can be useful for stylistic purposes or to wrap multiple paragraphs within a section that are all to be annotated in a similar way. In the following example, we see `<div>` elements used as a way to set the language of two paragraphs at once, instead of setting the language on the two paragraph elements separately:

```
<article lang="en-US">
  <h2>My use of language and my cats</h2>
  <p>My cat's behavior hasn't changed much since her absence, except
  that she plays her new physique to the neighbors regularly, in an
  attempt to get pets.</p>
  <div lang="en-GB">
    <p>My other cat, colored black and white, is a sweetie. He followed
    us to the pool today, walking down the pavement with us. Yesterday
    he apparently visited our neighbours. I wonder if he recognizes that
    their flat is a mirror image of ours.</p>
    <p>Hm, I just noticed that in the last paragraph I used British
    English. But I'm supposed to write in American English. So I
    shouldn't say "pavement" or "flat" or "color"...</p>
  </div>
  <p>I should say "sidewalk" and "apartment" and "color"!</p>
</article>
```

§ 4.5. Text-level semantics

§ 4.5.1. The `a` element**Categories:**[Flow content.](#)[Phrasing content.](#)If the element has an `href` attribute: [Interactive content](#).[Palpable content.](#)**Contexts in which this element can be used:**Where [phrasing content](#) is expected.

Content model:

[Transparent](#), but there must be no [interactive content](#) or `<a>` element descendants.

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

`href` - Address of the [hyperlink](#)

`target` - Default [browsing context](#) for [hyperlink navigation](#) and [§4.10.22 Form submission](#)

`download` - Whether to download the resource instead of navigating to it, and its file name if so

`rel` — Relationship of this document (or subsection/topic) to the destination resource

`rev` — [Reverse link](#) relationship of the destination resource to this document (or subsection/topic)

`hreflang` - Language of the linked resource

`type` - Hint for the type of the referenced resource

Allowed ARIA role attribute values:

`'link'` (default - *do not set*), `'button'`, `'checkbox'`, `'menuitem'`, `'menuitemcheckbox'`, `'menuitemradio'`, `'radio'`, `'switch'` [WAI-ARIA-1.1], `'tab'` or `'treeitem'`

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLAnchorElement : HTMLElement {
    attribute DOMString target;
    attribute DOMString download;
    attribute DOMString rel;
    attribute DOMString rev;
    [SameObject, PutForwards=value] readonly attribute DOMTokenList relList;
    attribute DOMString hreflang;
    attribute DOMString type;
    attribute DOMString text;
};
HTMLAnchorElement implements HTMLHyperlinkElementUtils;
```

If the `<a>` element has an `href` attribute, then it [represents](#) a [hyperlink](#) (a hypertext anchor) labeled by its contents.

If the `<a>` element has no `href` attribute, then the element [represents](#) a placeholder for where a link might otherwise have been placed, if it had been relevant, consisting of just the element's contents.

The `target`, `download`, `rel`, `rev`, `hreflang`, and `type` attributes must be omitted if the `href` attribute is not present.

EXAMPLE 172

If a site uses a consistent navigation toolbar on every page, then the link that would normally link to the page itself could be marked up using an `<a>` element:

```
<nav>
  <ul>
    <li> <a href="/">Home</a> </li>
    <li> <a href="/news">News</a> </li>
    <li> <a>Examples</a> </li>
    <li> <a href="/legal">Legal</a> </li>
  </ul>
</nav>
```

The `href`, `target`, `download`, and attributes affect what happens when users [follow hyperlinks](#) or [download hyperlinks](#) cre-

ated using the `<a>` element. The `rel`, `rev`, `hreflang`, and `type` attributes may be used to indicate to the user the likely nature of the target resource before the user follows the link.

The [activation behavior](#) of `<a>` elements that create [hyperlinks](#) is to run the following steps:

1. If the `<a>` element's [Document](#) is not [fully active](#), then abort these steps.
2. If either the `<a>` element has a `download` attribute and the algorithm is not [allowed to show a popup](#), or the element's `target` attribute is present and applying [the rules for choosing a browsing context given a browsing context name](#), using the value of the `target` attribute as the browsing context name, would result in there not being a chosen browsing context, then run these substeps:
 1. If there is an [entry settings object](#), throw an `InvalidAccessError` exception.
 2. Abort these steps without following the hyperlink.
3. If the target of the click event is an `<img>` element with an `ismap` attribute specified, then server-side image map processing must be performed, as follows:
 1. If the click event was a real pointing-device-triggered click event on the `<img>` element, then let x be the distance in CSS pixels from the left edge of the image's left border, if it has one, or the left edge of the image otherwise, to the location of the click, and let y be the distance in CSS pixels from the top edge of the image's top border, if it has one, or the top edge of the image otherwise, to the location of the click. Otherwise, let x and y be zero.
 2. Let *hyperlink suffix* be a U+003F QUESTION MARK character, the value of x expressed as a base-ten integer using [ASCII digits](#), a U+002C COMMA character (,), and the value of y expressed as a base-ten integer using [ASCII digits](#).
4. Finally, the user agent must [follow the hyperlink](#) or [download the hyperlink](#) created by the `<a>` element, as determined by the `download` attribute and any expressed user preference, passing *hyperlink suffix*, if the steps above defined it.

This definition is non-normative. Implementation requirements are given below this definition.

`a` . `text`

Same as [textContent](#).

The IDL attributes **`download`**, **`target`**, **`rel`**, **`rev`**, **`hreflang`**, and **`type`**, must [reflect](#) the respective content attributes of the same name.

The IDL attribute **`relList`** must [reflect](#) the [rel](#) content attribute.

The **`text`** IDL attribute, on getting, must return the same value as the [textContent](#) IDL attribute on the element, and on setting, must act as if the [textContent](#) IDL attribute on the element had been set to the new value.



The `<a>` element also supports the `HTMLHyperlinkElementUtils` interface. [\[URL\]](#)

When the element is created, and whenever the element's `href` content attribute is set, changed, or removed, the user agent must invoke the element's `HTMLHyperlinkElementUtils` interface's `set` the input algorithm with the value of the `href` content attribute, if any, or the empty string otherwise, as the given value.

The element's `HTMLHyperlinkElementUtils` interface's `get` the base algorithm must simply return the [document base URL](#).

The element's `HTMLHyperlinkElementUtils` interface's `query` encoding is the [document's character encoding](#).

When the element's `HTMLHyperlinkElementUtils` interface invokes its `update` steps with a string *value*, the user agent must set the element's `href` content attribute to the string *value*.

EXAMPLE 173

The `<a>` element may be wrapped around entire paragraphs, lists, tables, and so forth, even entire sections, so long as there is no interactive content within (e.g., buttons or other links). This example shows how this can be used to make an entire advertising block into a link:

```
<aside class="advertising">
  <h1>Advertising</h1>
  <a href="https://ad.example.com/?adid=1929&amp;pubid=1422">
    <section>
      <h1>Mellblomatic 9000!</h1>
      <p>Turn all your widgets into mellbloms!</p>
      <p>Only $9.99 plus shipping and handling.</p>
    </section>
  </a>
  <a href="https://ad.example.com/?adid=375&amp;pubid=1422">
    <section>
      <h1>The Mellblom Browser</h1>
      <p>Web browsing at the speed of light.</p>
      <p>No other browser goes faster!</p>
    </section>
  </a>
</aside>
```

4.5.2. The em element**Categories:**

[Flow content.](#)
[Phrasing content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[Any role value.](#)

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)
 Any aria-* attributes [applicable to the allowed roles.](#)

DOM interface:

Uses [HTMLElement](#).

The `<em>` element [represents](#) stress emphasis of its contents.

The level of stress that a particular piece of content has is given by its number of ancestor `<em>` elements.

The placement of stress emphasis changes the meaning of the sentence. The element thus forms an integral part of the content. The precise way in which stress is used in this way depends on the language.

EXAMPLE 174

These examples show how changing the stress emphasis changes the meaning. First, a general statement of fact, with no stress:

```
<p>Cats are cute animals.</p>
```

By emphasizing the first word, the statement implies that the kind of animal under discussion is in question (maybe someone is asserting that dogs are cute):

```
<p><em>Cats</em> are cute animals.</p>
```

Moving the stress to the verb, one highlights that the truth of the entire sentence is in question (maybe someone is saying cats are not cute):

```
<p>Cats <em>are</em> cute animals.</p>
```

By moving it to the adjective, the exact nature of the cats is reasserted (maybe someone suggested cats were *mean* animals):

```
<p>Cats are <em>cute</em> animals.</p>
```

Similarly, if someone asserted that cats were vegetables, someone correcting this might emphasize the last word:

```
<p>Cats are cute <em>animals</em>.</p>
```

By emphasizing the entire sentence, it becomes clear that the speaker is fighting hard to get the point across. This kind of stress emphasis also typically affects the punctuation, hence the exclamation mark here.

```
<p><em>Cats are cute animals!</em></p>
```

Anger mixed with emphasizing the cuteness could lead to markup such as:

```
<p><em>Cats are <em>cute</em> animals!</em></p>
```

The `<em>` element isn't a generic "italics" element. Sometimes, text is intended to stand out from the rest of the paragraph, as if it was in a different mood or voice. For this, the `<i>` element is more appropriate.

The `<em>` element also isn't intended to convey importance; for that purpose, the `<strong>` element is more appropriate.

§ 4.5.3. The strong element

Categories:

[Flow content.](#)

[Phrasing content.](#)

[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed [ARIA role attribute](#) values:

[Any role value.](#)

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles.](#)

DOM interface:

Uses [HTMLElement](#).

The `<strong>` element [represents](#) strong importance, seriousness, or urgency for its contents.

Importance: The `<strong>` element can be used in a heading, caption, or paragraph to distinguish the part that really matters from other parts that might be more detailed, more jovial, or merely boilerplate.

EXAMPLE 175

For example, the first word of the previous paragraph is marked up with `strong` to distinguish it from the more detailed text in the rest of the paragraph.

Seriousness: The `<strong>` element can be used to mark up a warning or caution notice.

Urgency: The `<strong>` element can be used to denote contents that the user needs to see sooner than other parts of the document.

The relative level of importance of a piece of content is given by its number of ancestor `<strong>` elements; each `<strong>` element increases the importance of its contents.

Changing the importance of a piece of text with the `<strong>` element does not change the meaning of the sentence.

EXAMPLE 176

Here, the word "chapter" and the actual chapter number are mere boilerplate, and the actual name of the chapter is marked up with `strong`:

```
<h1>Chapter 1: <strong>The Praxis</strong></h1>
```

In the following example, the name of the diagram in the caption is marked up with `<strong>`, to distinguish it from boilerplate text (before) and the description (after):

```
<figcaption>Figure 1. <strong>Ant colony dynamics</strong>. The ants in this colony  
are affected by the heat source (upper left) and the food source (lower right).</figcaption>
```

In this example, the heading is really "Flowers, Bees, and Honey", but the author has added a light-hearted addition to the heading. The `<strong>` element is thus used to mark up the first part to distinguish it from the latter part.

```
<h1><strong>Flowers, Bees, and Honey</strong> and other things I don't understand</h1>
```

EXAMPLE 177

Here is an example of a warning notice in a game, with the various parts marked up according to how important they are:

```
<p><strong>Warning.</strong> This dungeon is dangerous.  
<strong>Avoid the ducks.</strong> Take any gold you find.  
<strong><strong>Do not take any of the diamonds</strong>,&br/>they are explosive and <strong>will destroy anything within  
ten meters.</strong></strong> You have been warned.</p>
```

EXAMPLE 178

In this example, the `<strong>` element is used to denote the part of the text that the user is intended to read first.

```
<p>Welcome to Remy, the reminder system.</p>
<p>Your tasks for today:</p>
<ul>
  <li><p><strong>Turn off the oven.</strong></p></li>
  <li><p>Put out the trash.</p></li>
  <li><p>Do the laundry.</p></li>
</ul>
```

§ **4.5.4. The small element****Categories:**

[Flow content.](#)
[Phrasing content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[Any role value.](#)

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)
 Any aria-* attributes [applicable to the allowed roles.](#)

DOM interface:

Uses [HTMLElement](#).

The `<small>` element [represents](#) side comments such as small print.

NOTE:

Small print typically features disclaimers, caveats, legal restrictions, or copyrights. Small print is also sometimes used for attribution, or for satisfying licensing requirements.

NOTE:

The `<small>` element does not "de-emphasize" or lower the importance of text emphasized by the `<em>` element or marked as important with the strong element. To mark text as not emphasized or important, simply do not mark it up with the em or `<strong>` elements respectively.

The `<small>` element should not be used for extended spans of text, such as multiple paragraphs, lists, or sections of text. It is only intended for short runs of text. The text of a page listing terms of use, for instance, would not be a suitable candidate for the `<small>` element: in such a case, the text is not a side comment, it is the main content of the page.

EXAMPLE 179

In this example, the `<small>` element is used to indicate that value-added tax is not included in a price of a hotel room:

```
<dl>
  <dt>Single room
  <dd>199 € <small>breakfast included, VAT not included</small>
  <dt>Double room
  <dd>239 € <small>breakfast included, VAT not included</small>
</dl>
```

EXAMPLE 180

In this second example, the `<small>` element is used for a side comment in an article.

```
<p>Example Corp today announced record profits for the
second quarter <small>(Full Disclosure: Foo News is a subsidiary of
Example Corp)</small>, leading to speculation about a third quarter
merger with Demo Group.</p>
```

This is distinct from a sidebar, which might be multiple paragraphs long and is removed from the main flow of text. In the following example, we see a sidebar from the same article. This sidebar also has small print, indicating the source of the information in the sidebar.

```
<aside>
  <h1>Example Corp</h1>
  <p>This company mostly creates small software and Web
sites.</p>
  <p>The Example Corp company mission is "To provide entertainment
and news on a sample basis".</p>
  <p><small>Information obtained from <a
href="https://example.com/about.html">example.com</a> home
page.</small></p>
</aside>
```

In this last example, the `<small>` element is marked as being *important* small print.

```
<p><strong><small>Continued use of this service will result in a kiss.</small></strong></p>
```

§ 4.5.5. The `s` element**Categories:**

[Flow content.](#)
[Phrasing content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[Any role value.](#)

Allowed ARIA state and property attributes:[Global aria-* attributes](#)Any aria-* attributes [applicable to the allowed roles](#).**DOM interface:**Uses [HTMLElement](#).

The `<s>` element [represents](#) contents that are no longer accurate or no longer relevant.

NOTE:

The `<s>` element is not appropriate when indicating document edits; to mark a span of text as having been removed from a document, use the `<del>` element.

EXAMPLE 181

In this example a recommended retail price has been marked as no longer relevant as the product in question has a new sale price.

```
<p>Buy our Iced Tea and Lemonade!</p>
<p><s>Recommended retail price: $3.99 per bottle</s></p>
<p><strong>Now selling for just $2.99 a bottle!</strong></p>
```

4.5.6. The cite element**Categories:**[Flow content.](#)[Phrasing content.](#)[Palpable content.](#)**Contexts in which this element can be used:**Where [phrasing content](#) is expected.**Content model:**[Phrasing content.](#)**Tag omission in text/html:**

Neither tag is omissible

Content attributes:[Global attributes](#)**Allowed ARIA role attribute values:**[Any role value.](#)**Allowed ARIA state and property attributes:**[Global aria-* attributes](#)Any aria-* attributes [applicable to the allowed roles](#).**DOM interface:**Uses [HTMLElement](#).

The `<cite>` element [represents](#) a reference to a creative work. It must include the title of the work or the name of the author (person, people or organization) or an URL reference, or a reference in abbreviated form as per the conventions used for the addition of citation metadata.

NOTE:

Creative works include a book, a paper, an essay, a poem, a score, a song, a script, a film, a TV show, a game, a sculpture, a painting, a theatre production, a play, an opera, a musical, an exhibition, a legal case report, a computer program, , a web site, a web page, a blog post or comment, a forum post or comment, a tweet, a written or oral statement, etc.

EXAMPLE 182

Here is an example of the author of a quote referenced using the `<cite>` element:

```
<p>In the words of <cite>Charles Bukowski</cite> -
<q>An intellectual says a simple thing in a hard way. An artist says a hard thing in a simple
way.</q></p>
```

EXAMPLE 183

This second example identifies the author of a tweet by referencing the authors name using the `<cite>` element:

```
<blockquote class="twitter-tweet">
<p>♥ Bukowski in <a href="https://twitter.com/search?q=%23HTML5&src=hash">#HTML5</a> spec
examples
<a href="https://t.co/0FIEiYN1pC">https://t.co/0FIEiYN1pC</a></p><cite>- karl dubost
(@karlpro)
<a href="https://twitter.com/karlpro/statuses/370905307293442048">August 23, 2013</a></cite>
</blockquote>
```

EXAMPLE 184

In this example the `<cite>` element is used to reference the title of a work in a bibliography:

```
<p><cite>Universal Declaration of Human Rights</cite>, United Nations,
December 1948. Adopted by General Assembly resolution 217 A (III).</p>
```

EXAMPLE 185

In this example the `<cite>` element is used to reference the title of a television show:

```
<p>Who is your favorite doctor (in <cite>Doctor Who</cite>)?</p>
```

EXAMPLE 186

A very common use for the `<cite>` element is to identify the author of a comment in a blog post or forum, as in this example:

```
<article id="comment-1">
  Comment by <cite><a href="https://oli.jp">Oli Studholme</a></cite>
  <time datetime="2013-08-19T16:01">August 19th, 2013 at 4:01 pm</time>
  <p>Unfortunately I don't think adding names back into the definition of <code>cite</code>
  solves the problem: of the 12 blockquote examples in
  <a href="https://oli.jp/example/blockquote-metadata/">Examples of block quote metadata</a>,
  there's not even one that's <em>just</em> a person's name.</p>
  <p>A subset of the problem, maybe...</p>
</article>
```

EXAMPLE 187

Another common use for the `<cite>` element is to reference the URL of a search result, as in this example:

```
<div id="resultStats">About 416,000,000 results 0.33 seconds) </div>
...
<p><a href="https://www.w3.org/html/wg/">W3C <i>HTML Working Group</i></a></p>
<p><cite>www.w3.org/<b>html</b>/wg/</cite></p>
<p>15 Apr 2013 - The <i>HTML Working Group</i> is currently chartered to continue its
work through 31 December 2014. A Plan 2014 document published by the...</p>
...
```

EXAMPLE 188

Where the `<cite>` element is used to identify an abbreviated reference such as *Ibid.* it is suggested that this reference be linked to the base reference:

```
<article>
  <h2>Book notes</h2>
  ...
  ...
  <blockquote>"Money is the real cause of poverty,"
<footer>
  <cite id="baseref">The Ragged-Trousered Philanthropists, page 89.</cite>
</footer>
</blockquote>
  ...
  ...
  <blockquote>"Money is the cause of poverty because it is the device by which those who
are too lazy to work are enabled to rob the workers of the fruits of their labour."
  <a href="#baseref"><cite>Ibid.</cite></a>
</blockquote>
  ...
</article>
```

NOTE:

A citation is not a quote (for which the `<q>` element is appropriate).

EXAMPLE 189

This is incorrect usage, because `cite` is not for quotes:

```
<p><cite>This is wrong!, said Hillary.</cite> is a quote from the
popular daytime TV drama When Ian became Hillary.</p>
```

This is an example of the correct usage:

```
<p><q>This is correct, said Hillary.</q> is a quote from the
popular daytime TV drama <cite>When Ian became Hillary</cite>.</p>
```

§ 4.5.7. The q element**Categories:**

[Flow content.](#)
[Phrasing content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

`cite` - Link to the source of the quotation or more information about the edit

Allowed ARIA role attribute values:

[Any role value.](#)

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

Uses [HTMLQuoteElement](#).

The `<q>` element [represents](#) some [phrasing content](#) quoted from another source.

Quotation punctuation (such as quotation marks) that is quoting the contents of the element must not appear immediately before, after, or inside `<q>` elements; they will be inserted into the rendering by the user agent.

Content inside a `<q>` element must be quoted from another source, whose address, if it has one, may be cited in the `cite` attribute. The source may be fictional, as when quoting characters in a novel or screenplay.

If the `cite` attribute is present, it must be a [valid URL potentially surrounded by spaces](#). To obtain the corresponding citation link, the value of the attribute must be [parsed](#) relative to the element's [node document](#). User agents may allow users to follow such citation links, but they are primarily intended for private use (e.g., by server-side scripts collecting statistics about a site's use of quotations), not for readers.

The `<q>` element must not be used in place of quotation marks that do not represent quotes; for example, it is inappropriate to use the `<q>` element for marking up sarcastic statements.

The use of `<q>` elements to mark up quotations is entirely optional; using explicit quotation punctuation without `<q>` elements is just as correct.

EXAMPLE 190

Here is a simple example of the use of the `<q>` element:

```
<p>The man said <q>Things that are impossible just take  
longer</q>. I disagreed with him.</p>
```

EXAMPLE 191

Here is an example with both an explicit citation link in the `<q>` element, and an explicit citation outside:

```
<p>The W3C page <cite>About W3C</cite> says the W3C's  
mission is <q cite="https://www.w3.org/Consortium/">To lead the  
World Wide Web to its full potential by developing protocols and  
guidelines that ensure long-term growth for the Web</q>. I  
disagree with this mission.</p>
```

EXAMPLE 192

In the following example, the quotation itself contains a quotation:

```
<p>In <cite>Example One</cite>, he writes <q>The man  
said <q>Things that are impossible just take longer</q>. I  
disagreed with him</q>. Well, I disagree even more!</p>
```

EXAMPLE 193

In the following example, quotation marks are used instead of the `<q>` element:

```
<p>His best argument was "I disagree", which  
I thought was laughable.</p>
```

EXAMPLE 194

In the following example, there is no quote — the quotation marks are used to name a word. Use of the `<q>` element in this case would be inappropriate.

```
<p>The word "ineffable" could have been used to describe the disaster
resulting from the campaign's mismanagement.</p>
```

§ **4.5.8. The `dfn` element****Categories:**

[Flow content](#),
[Phrasing content](#),
[Palpable content](#).

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content](#), but there must be no `<dfn>` element descendants.

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Also, the [title](#) attribute has special semantics on this element.

Allowed ARIA role attribute values:

[Any role value](#).

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

Uses [HTMLElement](#).

The `<dfn>` element [represents](#) the defining instance of a term. The `<paragraph>`, `<description list group>`, or `<section>` that is the nearest ancestor of the `<dfn>` element must also contain the definition(s) for the [term](#) given by the `<dfn>` element.

Defining term: If the `<dfn>` element has a **title** attribute, then the exact value of that attribute is the term being defined. Otherwise, if it contains exactly one element child node and no child Text nodes, and that child element is an `<abbr>` element with a **title** attribute, then the exact value of *that* attribute is the term being defined. Otherwise, it is the exact [textContent](#) of the `<dfn>` element that gives the term being defined.

If the **title** attribute of the `<dfn>` element is present, then it must contain only the term being defined.

NOTE:

The **title** attribute of ancestor elements does not affect `<dfn>` elements.

An `<a>` element that links to a `<dfn>` element represents an instance of the term defined by the `<dfn>` element.

EXAMPLE 195

In the following fragment, the term "Garage Door Opener" is first defined in the first paragraph, then used in the second. In both cases, its abbreviation is what is actually displayed.

```
<p>The <dfn><abbr title="Garage Door Opener">GDO</abbr></dfn>
is a device that allows off-world teams to open the iris.</p>
<!-- ... later in the document: -->
<p>Teal'c activated his <abbr title="Garage Door Opener">GDO</abbr>
and so Hammond ordered the iris to be opened.</p>
```

With the addition of an [<a>](#) element, the reference can be made explicit:

```
<p>The <dfn id=gdo><abbr title="Garage Door Opener">GDO</abbr></dfn>
is a device that allows off-world teams to open the iris.</p>
<!-- ... later in the document: -->
<p>Teal'c activated his <a href=#gdo><abbr title="Garage Door Opener">GDO</abbr></a>
and so Hammond ordered the iris to be opened.</p>
```

4.5.9. The abbr element**Categories:**

[Flow content.](#)
[Phrasing content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Also, the [title](#) attribute has special semantics on this element.

Allowed ARIA role attribute values:

[Any role value.](#)

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles.](#)

DOM interface:

Uses [HTMLElement](#).

The [<abbr>](#) element [represents](#) an abbreviation or acronym, optionally with its expansion. The **title** attribute may be used to provide an expansion of the abbreviation. The attribute, if specified, must contain an expansion of the abbreviation, and nothing else.

EXAMPLE 196

The paragraph below contains an abbreviation marked up with the `<abbr>` element. This paragraph [defines the term](#) "Web Hypertext Application Technology Working Group".

```
<p>The <dfn id=whatwg><abbr
title="Web Hypertext Application Technology Working Group">WHATWG</abbr></dfn>
is a loose unofficial collaboration of Web browser manufacturers and
interested parties who wish to develop new technologies designed to
allow authors to write and deploy Applications over the World Wide
Web.</p>
```

An alternative way to write this would be:

```
<p>The <dfn id=whatwg>Web Hypertext Application Technology
Working Group</dfn> (<abbr
title="Web Hypertext Application Technology Working Group">WHATWG</abbr>)
is a loose unofficial collaboration of Web browser manufacturers and
interested parties who wish to develop new technologies designed to
allow authors to write and deploy Applications over the World Wide
Web.</p>
```

EXAMPLE 197

This paragraph has two abbreviations. Notice how only one is defined; the other, with no expansion associated with it, does not use the `<abbr>` element.

```
<p>The
<abbr title="Web Hypertext Application Technology Working Group">WHATWG</abbr>
started working on HTML in 2004.</p>
```

EXAMPLE 198

This paragraph links an abbreviation to its definition.

```
<p>The <a href="#whatwg"><abbr
title="Web Hypertext Application Technology Working Group">WHATWG</abbr></a>
community does not have much representation from Asia.</p>
```

EXAMPLE 199

This paragraph marks up an abbreviation without giving an expansion, possibly as a hook to apply styles for abbreviations (e.g., smallcaps).

```
<p>Philip and Dashiva both denied that they were going to
get the issue counts from past revisions of the specification to
backfill the <abbr>WHATWG</abbr> issue graph.</p>
```

If an abbreviation is pluralized, the expansion's grammatical number (plural vs singular) must match the grammatical number of the contents of the element.

EXAMPLE 200

Here the plural is outside the element, so the expansion is in the singular:

```
<p>Two <abbr title="Working Group">WG</abbr>s worked on
this specification: the <abbr>WHATWG</abbr> and the
<abbr>HTMLWG</abbr>.</p>
```

Here the plural is inside the element, so the expansion is in the plural:

```
<p>Two <abbr title="Working Groups">WGs</abbr> worked on
this specification: the <abbr>WHATWG</abbr> and the
<abbr>HTMLWG</abbr>.</p>
```

Abbreviations do not have to be marked up using this element. It is expected to be useful in the following cases:

- Abbreviations for which the author wants to give expansions, where using the `abbr` element with a `title` attribute is an alternative to including the expansion inline (e.g., in parentheses).
- Abbreviations that are likely to be unfamiliar to the document's readers, for which authors are encouraged to either mark up the abbreviation using an `abbr` element with a `title` attribute or include the expansion inline in the text the first time the abbreviation is used.
- Abbreviations whose presence needs to be semantically annotated, e.g., so that they can be identified from a style sheet and given specific styles, for which the `abbr` element can be used without a `title` attribute.

Providing an expansion in a `title` attribute once will not necessarily cause other `abbr` elements in the same document with the same contents but without a `title` attribute to behave as if they had the same expansion. Every `abbr` element is independent.

§ 4.5.10. The ruby element

Categories:

[Flow content.](#)
[Phrasing content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

See prose.

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[Any role value.](#)

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)
 Any `aria-*` attributes [applicable to the allowed roles](#).

DOM interface:

Uses [HTMLElement](#).

The `<ruby>` element allows one or more spans of phrasing content to be marked with ruby annotations. Ruby annotations are short runs of text presented alongside base text, primarily used in East Asian typography as a guide for pronunciation or to include other annotations. In Japanese, this form of typography is also known as *furigana*. Ruby text can appear on either

side, and sometimes both sides, of the base text, and it is possible to control its position using CSS. A more complete introduction to ruby can be found in the *Use Cases & Exploratory Approaches for Ruby Markup* document as well as in *CSS Ruby*. [\[RUBY-UC\]](#) [\[CSS3-RUBY\]](#)

The content model of `<ruby>` elements consists of one or more of the following sequences:

1. One or more [phrasing content](#) nodes or `<rb>` elements.
2. One or more `rt` or `<rtc>` elements, each of which either immediately preceded or followed by an `<rp>` elements.

The `<ruby>`, `<rb>`, `<rtc>`, and `<rt>` elements can be used for a variety of kinds of annotations, including in particular (though by no means limited to) those described below. For more details on Japanese Ruby in particular, and how to render Ruby for Japanese, see *Requirements for Japanese Text Layout*. [\[JLREQ\]](#) The `rp` element can be used as fallback content when ruby rendering is not supported.

Mono-ruby for individual base characters

Annotations (the ruby text) are associated individually with each ideographic character (the base text). In Japanese this is typically hiragana or katakana characters used to provide readings of kanji characters.

EXAMPLE 201

```
<ruby>base<rt>annotation</ruby>
```

When no `<rb>` element is used, the base is implied, as above. But you can also make it explicit. This can be useful notably for styling, or when consecutive bases are to be treated as a group, as in the *jukugo* ruby example further down.

EXAMPLE 202

```
<ruby><rb>base<rt>annotation</ruby>
```

In the following example, notice how each annotation corresponds to a single base character.

EXAMPLE 203

```
<ruby>日<rt>に</rt></ruby><ruby>本<rt>ほん</rt></ruby>
<ruby>語<rt>ご</rt></ruby>で<ruby>書<rt>か</rt></ruby>
いた<ruby>作<rt>さく</rt></ruby><ruby>文<rt>ぶん</rt></ruby>です。
```

Ruby text interspersed in regular text provides structure akin to the following image:

にほんご か さくぶん
日本語で書いた作文です。

This example can also be written as follows, using one `<ruby>` element with two segments of base text and two annotations (one for each) rather than two back-to-back `<ruby>` elements each with one base text segment and annotation (as in the markup above):

EXAMPLE 204

```
<ruby>日<rt>に</rt>本<rt>ほん</rt>語<rt>ご</rt></ruby>
で<ruby>書<rt>か</rt></ruby>
いた<ruby>作<rt>さく</rt>文<rt>ぶん</rt></ruby>です。
```

Group ruby

Group ruby is often used where phonetic annotations don't map to discreet base characters, or for semantic glosses that span the whole base text. For example, the word "today" is written with the characters 今日, literally "this

day". But it's pronounced きょう (kyou), which can't be broken down into a "this" part and a "day" part. In typical rendering, you can't split text that is annotated with group ruby; it has to wrap as a single unit onto the next line. When a [ruby text annotation](#) maps to a base that is comprised of more than one character, then that base is grouped.

The following group ruby:



Can be marked up as follows:

EXAMPLE 205

```
<ruby>今日<rt>きょう</ruby>
```

Jukugo ruby

Jukugo refers to a Japanese compound noun, i.e., a word made up of more than one kanji character. *Jukugo ruby* is a term that is used not to describe ruby annotations over jukugo text, but rather to describe ruby with a behavior slightly different from mono or group ruby. Jukugo ruby is similar to mono ruby, in that there is a strong association between ruby text and individual base characters, but the ruby text is typically rendered as grouped together over multiple ideographs when they are on the same line.

The distinction is captured in this example:



Which can be marked up as follows:

EXAMPLE 206

```
<ruby>法<rb>華<rb>経<rt>ほ<rt>け<rt>きょう</ruby>
```

In this example, each `<rt>` element is paired with its respective `rb` element, the difference with an interleaved `rb/rt` approach being that the sequences of both base text and ruby annotations are implicitly placed in common containers so that the grouping information is captured.

NOTE:

For more details on [Jukugo Ruby rendering](#), see Appendix F in the *Requirements for Japanese Text Layout* and Use Case C: Jukugo ruby in the *Use Cases & Exploratory Approaches for Ruby Markup*. [\[JLREQ\]](#)
[\[RUBY-UC\]](#)

Inline ruby

In some contexts, for instance when the font size or line height are too small for ruby to be readable, it is desirable to inline the ruby annotation such that it appears in parentheses after the text it annotates. This also provides a convenient fallback strategy for user agents that do not support rendering ruby annotations.

Inlining takes grouping into account. For example, Tokyo is written with two kanji characters, 東, which is pronounced と, and 京, which is pronounced きょう. Each base character should be annotated individually, but the fallback should be 東京(とうきょう) not 東(とう)京(きょう). This can be marked up as follows:

EXAMPLE 207

```
<ruby>東<rb>京<rt>とう<rt>きょう</ruby>
```

Note that the above markup will enable the usage of parentheses when inlining for browsers that support ruby layout, but for those that don't it will fail to provide parenthetical fallback. This is where the `<rp>` element is useful. It can be inserted into the above example to provide the appropriate fallback when ruby layout is not supported:

EXAMPLE 208

```
<ruby>東<rb>京<rp>(<rt>とう<rt>きょう<rp>)</ruby>
```

Text with both phonetic and semantic annotations (double-sided ruby)

Sometimes, ruby can be used to annotate a base twice.

In the following example, the Chinese word for San Francisco (旧金山, i.e., "old gold mountain") is annotated both using pinyin to give the pronunciation, and with the original English.



Which is marked up as follows:

EXAMPLE 209

```
<ruby><rb>旧<rb>金<rb>山<rt>jiù<rt>jīn<rt>shān<rtc>San Francisco</ruby>
```

In this example, a single base run of three base characters is annotated with three pinyin ruby text segments in a first (implicit) container, and an `<rtc>` element is introduced in order to provide a second single [ruby text annotation](#) being the city's English name.

We can also revisit our *jukugo* example above with 上手 ("skill") to show how it can be annotation in both kana and romaji phonetics while at the same time maintaining the pairing to bases and annotation grouping information.



Which is marked up as follows:

EXAMPLE 210

```
<ruby><rb>上<rb>手<rt>じょう<rt>ず<rtc><rt>jou<rt>zu</ruby>
```

Text that is a direct child of the `<rtc>` element implicitly produces a ruby text segment as if it were contained in an `<rt>` element. In this contrived example, this is shown with some symbols that are given names in English and French with annotations intended to appear on either side of the base symbol.

EXAMPLE 211

```

<ruby>
  ♥<rt>Heart<rtc lang=fr>Cœur</rtc>
  ♣<rt>Shamrock<rtc lang=fr>Trèfle</rtc>
  ★<rt>Star<rtc lang=fr>Étoile
</ruby>

```

Similarly, text directly inside a [<ruby>](#) element implicitly produces a ruby base as if it were contained in an [<rb>](#) element, and [rt](#) children of ruby are implicitly contained in an [rtc](#) container. In effect, the above example is equivalent (in meaning, though not in the DOM it produces) to the following:

EXAMPLE 212

```

<ruby>
  <rb>♥</rb><rtc><rt>Heart</rt></rtc><rtc lang=fr><rt>Cœur</rt></rtc>
  <rb>♣</rb><rtc><rt>Shamrock</rt></rtc><rtc lang=fr><rt>Trèfle</rt></rtc>
  <rb>★</rb><rtc><rt>Star</rt></rtc><rtc lang=fr><rt>Étoile</rt></rtc>
</ruby>

```



Within a ruby element, content is parcelled into a series of ruby segments. Each **ruby segment** is described by:

- Zero or more **ruby bases**, each of which is a DOM range that may contain phrasing content or an [<rb>](#) element.
- A base range, that is a DOM range including all the bases. This is the **ruby base container**.
- Zero or more [ruby text containers](#) which may correspond to explicit [rtc](#) elements, or to sequences of [<rt>](#) elements implicitly recognized as contained in an anonymous [ruby text container](#).

Each **ruby text container** is described by zero or more **ruby text annotations** each of which is a DOM range that may contain phrasing content or an [<rt>](#) element, and an annotations range that is a range including all the annotations for that container. A [ruby text container](#) is also known (primarily in a CSS context) as a **ruby annotation container**.

Furthermore, a ruby element contains **ignored ruby content**. Ignored ruby content does not form part of the document's semantics. It consists of some [inter-element whitespace](#) and [<rp>](#) elements, the latter of which are used for legacy user agents that do not support ruby at all.

The process of **annotation pairing** associates [ruby annotations](#) with [ruby bases](#). Within each [ruby segment](#), each [ruby base](#) in the [ruby base container](#) is paired with one [ruby text annotation](#) from the [ruby text container](#), in order. If there are not enough [ruby text annotations](#) in a [ruby annotation container](#), the last one is associated with any excess [ruby bases](#). (If there are not any in the [ruby annotation container](#), an anonymous empty one is assumed to exist.) If there are not enough [ruby bases](#), any remaining [ruby text annotations](#) are assumed to be associated with empty, anonymous bases inserted at the end of the [ruby base container](#).

Note that the terms [ruby segment](#), [ruby base](#), [ruby text annotation](#), [ruby text container](#), [ruby base container](#), and [ruby annotation container](#) have their equivalents in *CSS Ruby Module Level 3*. [\[CSS3-RUBY\]](#)

Informally, the segmentation and categorization algorithm below performs a simple set of tasks. First it processes adjacent [<rb>](#) elements, text nodes, and non-ruby elements into a list of bases. Then it processes any number of [<rtc>](#) elements or sequences of [<rt>](#) elements that are considered to automatically map to an anonymous [ruby text container](#). Put together these data items form a [ruby segment](#) as detailed in the data model above. It will continue to produce such segments until it reaches the end of the content of a given [<ruby>](#) element. The complexity of the algorithm below compared to this informal description stems from the need to support an author-friendly syntax and being mindful of inter-element white space.

At any particular time, the **segmentation and categorization of content of a ruby element** is the result that would be obtained from running the following algorithm:

1. Let *root* be the [<ruby>](#) element for which the algorithm is being run.

2. Let *index* be 0.
3. Let *ruby segments* be an empty list.
4. Let *current bases* be an empty list of DOM ranges.
5. Let *current bases range* be null.
6. Let *current bases range start* be null.
7. Let *current annotations* be an empty list of DOM ranges.
8. Let *current annotations range* be null.
9. Let *current annotations range start* be null.
10. Let *current annotation containers* be an empty list.
11. Let *current automatic base nodes* be an empty list of DOM Nodes.
12. Let *current automatic base range start* be null.
13. *Process a ruby child*: If *index* is equal to or greater than the number of child nodes in *root*, then run the steps to [commit a ruby segment](#), return *ruby segments*, and abort these steps.
14. Let *current child* be the *index*th node in *root*.
15. If *current child* is not a [Text](#) node and is not an [Element](#) node, then increment *index* by one and jump to the step labelled *process a ruby child*.
16. If *current child* is an [<rp>](#) element, then increment *index* by one and jump to the step labelled *process a ruby child*.
(Note that this has the effect of including this element in any range that we are currently processing. This is done intentionally so that misplaced rp can be processed correctly; semantically they are ignored all the same.)
17. If *current child* is an [<rt>](#) element, then run these substeps:
 1. Run the steps to [commit an automatic base](#).
 2. Run the steps to [commit the base range](#).
 3. If *current annotations* is empty, set *current annotations range start* to the value of *index*.
 4. Create a new DOM range whose *start* is the [boundary point](#) (*root*, *index*) and whose *end* is the [boundary point](#) (*root*, *index* plus one), and append it at the end of *current annotations*.
 5. Increment *index* by one and jump to the step labelled *process a ruby child*.
18. If *current child* is an [<rtc>](#) element, then run these substeps:
 1. Run the steps to [commit an automatic base](#).
 2. Run the steps to [commit the base range](#).
 3. Run the steps to [commit current annotations](#).
 4. Create a new [ruby annotation container](#). It is described by the list of annotations returned by running the steps to [process an rtc element](#) and a DOM range whose *start* is the [boundary point](#) (*root*, *index*) and whose *end* is the [boundary point](#) (*root*, *index* plus one). Append this new [ruby annotation container](#) at the end of *current annotation containers*.
 5. Increment *index* by one and jump to the step labelled *process a ruby child*.
19. If *current child* is a [Text](#) node and is [inter-element whitespace](#), then run these substeps:
 1. If *current annotations* is not empty, increment *index* by one and jump to the step labelled *process a ruby child*.
 2. Run the following substeps:
 1. Let *lookahead index* be set to the value of *index*.
 2. *Peek ahead*: Increment *lookahead index* by one.
 3. If *lookahead index* is equal to or greater than the number of child nodes in *root*, then abort these substeps.
 4. Let *peek child* be the *lookahead index*th node in *root*.

5. If *peek child* is a [Text](#) node and is [inter-element whitespace](#), then jump to the step labelled *peek ahead*.
 6. If *peek child* is an [<rt>](#) element, an [<rtc>](#) element, or an [<rp>](#) element, then set *index* to the value of *lookahead index* and jump to the step labelled *process a ruby child*.
20. If *current annotations* is not empty or if *current annotation containers* is not empty, then run the steps to [commit a ruby segment](#).
 21. If *current child* is an [<rb>](#) element, then run these substeps:
 1. Run the steps to [commit an automatic base](#).
 2. If *current bases* is empty, then set *current bases range start* to the value of *index*.
 3. Create a new DOM range whose *start* is the [boundary point](#) (*root*, *index*) and whose *end* is the [boundary point](#) (*root*, *index* plus one), and append it at the end of *current bases*.
 4. Increment *index* by one and jump to the step labelled *process a ruby child*.
 22. If *current automatic base nodes* is empty, set *current automatic base range start* to the value of *index*.
 23. Append *current child* at the end of *current automatic base nodes*.
 24. Increment *index* by one and jump to the step labelled *process a ruby child*.

When the steps above say to **commit a ruby segment**, it means to run the following steps at that point in the algorithm:

1. Run the steps to [commit an automatic base](#).
2. If *current bases*, *current annotations*, and *current annotation containers* are all empty, abort these steps.
3. Run the steps to [commit the base range](#).
4. Run the steps to [commit current annotations](#).
5. Create a new [ruby segment](#). It is described by a list of bases set to *current bases*, a base DOM range set to *current bases range*, and a list of [ruby annotation containers](#) that are the *current annotation containers* list. Append this new [ruby segment](#) at the end of *ruby segments*.
6. Let *current bases* be an empty list.
7. Let *current bases range* be null.
8. Let *current bases range start* be null.
9. Let *current annotation containers* be an empty list.

When the steps above say to **commit the base range**, it means to run the following steps at that point in the algorithm:

1. If *current bases* is empty, abort these steps.
2. If *current bases range* is not null, abort these steps.
3. Let *current bases range* be a DOM range whose *start* is the [boundary point](#) (*root*, *current bases range start*) and whose *end* is the [boundary point](#) (*root*, *index*).

When the steps above say to **commit current annotations**, it means to run the following steps at that point in the algorithm:

1. If *current annotations* is not empty and *current annotations range* is null let *current annotations range* be a DOM range whose *start* is the [boundary point](#) (*root*, *current annotations range start*) and whose *end* is the [boundary point](#) (*root*, *index*).
2. If *current annotations* is not empty, create a new [ruby annotation container](#). It is described by an annotations list set to *current annotations* and a range set to *current annotations range*. Append this new [ruby annotation container](#) at the end of *current annotation containers*.
3. Let *current annotations* be an empty list of DOM ranges.
4. Let *current annotations range* be null.

5. Let *current annotations range start* be null.

When the steps above say to **commit an automatic base**, it means to run the following steps at that point in the algorithm:

1. If *current automatic base nodes* is empty, abort these steps.
2. If *current automatic base nodes* contains nodes that are not [Text](#) nodes, or [Text](#) nodes that are not [inter-element whitespace](#), then run these substeps:
 1. If *current bases* is empty, set *current bases range start* to the value of *current automatic base range start*.
 2. Create a new DOM range whose [start](#) is the [boundary point](#) (*root*, *current automatic base range start*) and whose [end](#) is the [boundary point](#) (*root*, *index*), and append it at the end of *current bases*.
3. Let *current automatic base nodes* be an empty list of DOM Nodes.
4. Let *current automatic base range start* be null.

§ 4.5.11. The rb element

Categories:

None.

Contexts in which this element can be used:

As a child of a [<ruby>](#) element.

Content model:

[Phrasing content](#).

Tag omission in text/html:

An [<rb>](#) element's [end tag](#) may be omitted if the [<rb>](#) element is immediately followed by an [<rb>](#), [<rt>](#), [rtc](#) or [<rp>](#) element, or if there is no more content in the parent element.

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[Any role value](#).

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

Uses [HTMLElement](#).

The [<rb>](#) element marks the base text component of a ruby annotation. When it is the child of a [<ruby>](#) element, it doesn't [represent](#) anything itself, but its parent ruby element uses it as part of determining what *it* [represents](#).

An [<rb>](#) element that is not a child of a [<ruby>](#) element [represents](#) the same thing as its children.

§ 4.5.12. The rt element

Categories:

None.

Contexts in which this element can be used:

As a child of a ruby or of an [<rtc>](#) element.

Content model:

[Phrasing content](#).

Tag omission in text/html:

An [<rt>](#) element's [end tag](#) may be omitted if the [<rt>](#) element is immediately followed by an [<rb>](#), [<rt>](#), [rtc](#) or [<rp>](#) element, or if there is no more content in the parent element.

Content attributes:[Global attributes](#)**Allowed ARIA role attribute values:**[Any role value.](#)**Allowed ARIA state and property attributes:**[Global aria-* attributes](#)Any aria-* attributes [applicable to the allowed roles](#).**DOM interface:**Uses [HTMLElement](#).

The `<rt>` element marks the ruby text component of a ruby annotation. When it is the child of a `<ruby>` element or of an `<rtc>` element that is itself the child of a `<ruby>` element, it doesn't [represent](#) anything itself, but its ancestor ruby element uses it as part of determining what *it represents*.

An `<rt>` element that is not a child of a `<ruby>` element or of an `<rtc>` element that is itself the child of a `<ruby>` element [represents](#) the same thing as its children.

4.5.13. The rtc element**Categories:**

None.

Contexts in which this element can be used:As a child of a `<ruby>` element.**Content model:**[Phrasing content](#), `<rt>`, or `<rp>` elements.**Tag omission in text/html:**

An `<rtc>` element's [end tag](#) may be omitted if the `<rtc>` element is immediately followed by an `rb` or `<rtc>` element, or if there is no more content in the parent element.

Content attributes:[Global attributes](#)**Allowed ARIA role attribute values:**[Any role value.](#)**Allowed ARIA state and property attributes:**[Global aria-* attributes](#)Any aria-* attributes [applicable to the allowed roles](#).**DOM interface:**Uses [HTMLElement](#).

The `<rtc>` element marks a [ruby text container](#) for ruby text components in a ruby annotation. When it is the child of a `<ruby>` element it doesn't [represent](#) anything itself, but its parent `<ruby>` element uses it as part of determining what *it represents*.

An `<rtc>` element that is not a child of a `<ruby>` element [represents](#) the same thing as its children.

When an `<rtc>` element is processed as part of the segmentation and categorization of content for a `<ruby>` element, the following algorithm defines how to **process an `<rtc>` element**:

1. Let *root* be the `<rtc>` element for which the algorithm is being run.
2. Let *index* be 0.
3. Let *annotations* be an empty list of DOM ranges.
4. Let *current automatic annotation nodes* be an empty list of DOM nodes.

5. Let *current automatic annotation range start* be null.
6. *Process an rtc child*: If *index* is equal to or greater than the number of child nodes in *root*, then run the steps to [commit an automatic annotation](#), return *annotations*, and abort these steps.
7. Let *current child* be the *index*th node in *root*.
8. If *current child* is an `<rt>` element, then run these substeps:
 1. Run the steps to [commit an automatic annotation](#).
 2. Create a new DOM range whose *start* is the [boundary point](#) (*root*, *index*) and whose *end* is the [boundary point](#) (*root*, *index* plus one), and append it at the end of *annotations*.
 3. Increment *index* by one and jump to the step labelled *process an rtc child*.
9. If *current automatic annotation nodes* is empty, set *current automatic annotation range start* to the value of *index*.
10. Append *current child* at the end of *current automatic annotation nodes*.
11. Increment *index* by one and jump to the step labelled *process an rtc child*.

When the steps above say to **commit an automatic annotation**, it means to run the following steps at that point in the algorithm:

1. If *current automatic annotation nodes* is empty, abort these steps.
2. If *current automatic annotation nodes* contains nodes that are not [Text](#) nodes, or [Text](#) nodes that are not [inter-element whitespace](#), then create a new DOM range whose *start* is the [boundary point](#) (*root*, *current automatic annotation range start*) and whose *end* is the [boundary point](#) (*root*, *index*), and append it at the end of *annotations*.
3. Let *current automatic annotation nodes* be an empty list of DOM nodes.
4. Let *current automatic annotation range start* be null.

§ 4.5.14. The `rp` element

Categories:

None.

Contexts in which this element can be used:

As a child of a ruby or `<rtc>` element, either immediately before or immediately after an `rt` or `<rtc>` element, but not between `<rt>` elements.

Content model:

[Phrasing content](#).

Tag omission in text/html:

An `<rp>` element's *end tag* may be omitted if the `<rp>` element is immediately followed by an `<rb>`, `<rt>`, `rtc` or `<rp>` element, or if there is no more content in the parent element.

Content attributes:

[Global attributes](#)

Allowed [ARIA role attribute](#) values:

[Any role value](#).

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any `aria-*` attributes [applicable to the allowed roles](#).

DOM interface:

Uses [HTMLElement](#).

The `<rp>` element is used to provide fallback text to be shown by user agents that don't support ruby annotations. One widespread convention is to provide parentheses around the ruby text component of a ruby annotation.

The contents of the [<rp>](#) elements are typically not displayed by user agents which do support ruby annotations

An [<rp>](#) element that is a child of a ruby element [represents](#) nothing. An `rp` element whose parent element is not a [<ruby>](#) element [represents](#) its children.

The example shown previously, in which each ideograph in the text 漢字 is annotated with its phonetic reading, could be expanded to use `rp` so that in legacy user agents the readings are in parentheses (please note that white space has been introduced into this example in order to make it more readable):

EXAMPLE 213

```
...
<ruby>
  漢
  <rb>字</rb>
  <rp> (</rp>
  <rt>かん</rt>
  <rt>じ</rt>
  <rp>) </rp>
</ruby>
...
```

In conforming user agents the rendering would be as above, but in user agents that do not support ruby, the rendering would be:

EXAMPLE 214

... 漢字 (かんじ) ...

When there are multiple annotations for a segment, [<rp>](#) elements can also be placed between the annotations. Here is another copy of an earlier contrived example showing some symbols with names given in English and French using double-sided annotations, but this time with [<rp>](#) elements as well:

EXAMPLE 215

```
<ruby>
  ♥<rp>: </rp><rt>Heart</rt><rp>, </rp><rtc><rt lang=fr>Cœur</rt></rtc><rp>.</rp>
  ♣<rp>: </rp><rt>Shamrock</rt><rp>, </rp><rtc><rt lang=fr>Trèfle</rt></rtc><rp>.</rp>
  *<rp>: </rp><rt>Star</rt><rp>, </rp><rtc><rt lang=fr>Étoile</rt></rtc><rp>.</rp>
</ruby>
```

This would make the example render as follows in non-ruby-capable user agents:

EXAMPLE 216

♥: Heart, Cœur.
 ♣: Shamrock, Trèfle.
 *: Star, Étoile.

§ 4.5.15. The data element

Categories:

[Flow content.](#)
[Phrasing content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content](#).

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

value - Machine-readable value

Allowed [ARIA role attribute](#) values:

[Any role value](#).

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLDataElement : HTMLElement {
    attribute DOMString value;
};
```

The `<data>` element [represents](#) its contents, along with a machine-readable form of those contents in the value attribute.

The **value** attribute must be present. Its value must be a representation of the element's contents in a machine-readable format.

NOTE:

When the value is date- or time-related, the more specific `time` element can be used instead.

The element can be used for several purposes.

When combined with microformats or microdata, the element serves to provide both a machine-readable value for the purposes of data processors, and a human-readable value for the purposes of rendering in a Web browser. In this case, the format to be used in the value attribute is determined by the microformats or microdata vocabulary in use.

The element can also, however, be used in conjunction with scripts in the page, for when a script has a literal value to store alongside a human-readable value. In such cases, the format to be used depends only on the needs of the script. (The `data-*` attributes can also be useful in such situations.)

The **value** IDL attribute must [reflect](#) the content attribute of the same name.

§ 4.5.16. The `time` element

Categories:

[Flow content](#).

[Phrasing content](#).

[Palpable content](#).

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

If the element has a `datetime` attribute: [Phrasing content](#).

Otherwise: [Text](#), but must match requirements described in prose below.

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

`datetime` - Machine-readable value

Allowed [ARIA role attribute](#) values:

[Any role value.](#)

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles.](#)

[DOM interface](#):

```
interface HTMLTimeElement : HTMLElement {
  attribute DOMString dateTime;
};
```

The `<time>` element [represents](#) its contents, along with a machine-readable form of those contents in the `datetime` attribute. The kind of content is limited to various kinds of dates, times, time-zone offsets, and durations, as described below.

The `datetime` attribute may be present. If present, its value must be a representation of the element's contents in a machine-readable format.

A `<time>` element that does not have a `datetime` content attribute must not have any element descendants.

The **datetime value** of a `<time>` element is the value of the element's `datetime` content attribute, if it has one, otherwise the concatenation of the contents of all the [Text](#) nodes that are children of the `<time>` element (ignoring any other nodes such as comments or elements), in [tree order](#).

The [datetime value](#) of a `<time>` element must match one of the following syntaxes.

A valid month string

```
EXAMPLE 217
<time>2011-11</time>
```

A valid date string

```
EXAMPLE 218
<time>2011-11-18</time>
```

A valid yearless date string

```
EXAMPLE 219
<time>11-18</time>
```

A valid time string

```
EXAMPLE 220
<time>14:54</time>
```

```
EXAMPLE 221
<time>14:54:39</time>
```

```
EXAMPLE 222
<time>14:54:39.929</time>
```

A valid floating date and time string

```
EXAMPLE 223
<time>2011-11-18T14:54</time>
```

```
EXAMPLE 224
<time>2011-11-18T14:54:39</time>
```

EXAMPLE 225

```
<time>2011-11-18T14:54:39.929</time>
```

EXAMPLE 226

```
<time>2011-11-18 14:54</time>
```

EXAMPLE 227

```
<time>2011-11-18 14:54:39</time>
```

EXAMPLE 228

```
<time>2011-11-18 14:54:39.929</time>
```

NOTE:

Times with dates but without a time zone offset are useful for specifying events that are observed at the same specific time in each time zone, throughout a day. For example, the 2020 new year is celebrated at 2020-01-01 00:00 in each time zone, not at the same precise moment across all time zones. For events that occur at the same time across all time zones, for example a videoconference meeting, a [valid global date and time string](#) is likely more useful.

A [valid time-zone offset string](#)**EXAMPLE 229**

```
<time>Z</time>
```

EXAMPLE 230

```
<time>+0000</time>
```

EXAMPLE 231

```
<time>+00:00</time>
```

EXAMPLE 232

```
<time>-0800</time>
```

EXAMPLE 233

```
<time>-08:00</time>
```

NOTE:

For times without dates (or times referring to events that recur on multiple dates), specifying the geographic location that controls the time is usually more useful than specifying a time zone offset, because geographic locations change time zone offsets with daylight savings time. In some cases, geographic locations even change time zone, e.g., when the boundaries of those time zones are redrawn, as happened with Samoa at the end of 2011. There exists a time zone database that describes the boundaries of time zones and what rules apply within each such zone, known as the time zone database. [\[TZDATABASE\]](#)

A [valid global date and time string](#)**EXAMPLE 234**

```
<time>2011-11-18T14:54Z</time>
```

EXAMPLE 235

```
<time>2011-11-18T14:54:39Z</time>
```

EXAMPLE 236

```
<time>2011-11-18T14:54:39.929Z</time>
```

EXAMPLE 237

```
<time>2011-11-18T14:54+0000</time>
```


EXAMPLE 238

```
<time>2011-11-18T14:54:39+0000</time>
```

EXAMPLE 239

```
<time>2011-11-18T14:54:39.929+0000</time>
```

EXAMPLE 240

```
<time>2011-11-18T14:54+00:00</time>
```

EXAMPLE 241

```
<time>2011-11-18T14:54:39+00:00</time>
```

EXAMPLE 242

```
<time>2011-11-18T14:54:39.929+00:00</time>
```

EXAMPLE 243

```
<time>2011-11-18T06:54-0800</time>
```

EXAMPLE 244

```
<time>2011-11-18T06:54:39-0800</time>
```

EXAMPLE 245

```
<time>2011-11-18T06:54:39.929-0800</time>
```

EXAMPLE 246

```
<time>2011-11-18T06:54-08:00</time>
```

EXAMPLE 247

```
<time>2011-11-18T06:54:39-08:00</time>
```

EXAMPLE 248

```
<time>2011-11-18T06:54:39.929-08:00</time>
```

EXAMPLE 249

```
<time>2011-11-18 14:54Z</time>
```

EXAMPLE 250

```
<time>2011-11-18 14:54:39Z</time>
```

EXAMPLE 251

```
<time>2011-11-18 14:54:39.929Z</time>
```

EXAMPLE 252

```
<time>2011-11-18 14:54+0000</time>
```

EXAMPLE 253

```
<time>2011-11-18 14:54:39+0000</time>
```

EXAMPLE 254

```
<time>2011-11-18 14:54:39.929+0000</time>
```

EXAMPLE 255

```
<time>2011-11-18 14:54+00:00</time>
```

EXAMPLE 256

```
<time>2011-11-18 14:54:39+00:00</time>
```

EXAMPLE 257`<time>2011-11-18 14:54:39.929+00:00</time>`**EXAMPLE 258**`<time>2011-11-18 06:54-0800</time>`**EXAMPLE 259**`<time>2011-11-18 06:54:39-0800</time>`**EXAMPLE 260**`<time>2011-11-18 06:54:39.929-0800</time>`**EXAMPLE 261**`<time>2011-11-18 06:54-08:00</time>`**EXAMPLE 262**`<time>2011-11-18 06:54:39-08:00</time>`**EXAMPLE 263**`<time>2011-11-18 06:54:39.929-08:00</time>`**NOTE:**

Times with dates and a time zone offset are useful for specifying specific events, or recurring virtual events where the time is not anchored to a specific geographic location. For example, the precise time of an asteroid impact, or a particular meeting in a series of meetings held at 1400 UTC every day, regardless of whether any particular part of the world is observing daylight savings time or not. For events where the precise time varies by the local time zone offset of a specific geographic location, a [valid floating date and time string](#) combined with that geographic location is likely more useful.

A [valid week string](#)**EXAMPLE 264**`<time>2011-W47</time>`

Four or more [ASCII digits](#), at least one of which is not U+0030 DIGIT ZERO (0)

EXAMPLE 265`<time>2011</time>`**EXAMPLE 266**`<time>0001</time>`**A [valid duration string](#)****EXAMPLE 267**`<time>PT4H18M3S</time>`**EXAMPLE 268**`<time>4h 18m 3s</time>`**NOTE:**

Many of the preceding valid syntaxes describe "floating" date and/or time values (they do not include a [time-zone offset](#)). Care is needed when converting floating time values to or from global ("incremental") time values (e.g., JavaScript's Date object). In many cases, an implicit time-of-day and time zone are used in the conversion and may result in unexpected changes to the value of the date itself. [\[TIMEZONE\]](#)

The **machine-readable equivalent of the element's contents** must be obtained from the element's [datetime value](#) by using the following algorithm:

1. If [parsing a month string](#) from the element's [datetime value](#) returns a [month](#), that is the machine-readable equivalent; abort these steps.
2. If [parsing a date string](#) from the element's [datetime value](#) returns a [date](#), that is the machine-readable equivalent; abort these steps.
3. If [parsing a yearless date string](#) from the element's [datetime value](#) returns a [yearless date](#), that is the machine-readable equivalent; abort these steps.
4. If [parsing a time string](#) from the element's [datetime value](#) returns a [time](#), that is the machine-readable equivalent; abort these steps.
5. If [parsing a floating date and time string](#) from the element's [datetime value](#) returns a [floating date and time](#), that is the machine-readable equivalent; abort these steps.
6. If [parsing a time-zone offset string](#) from the element's [datetime value](#) returns a [time-zone offset](#), that is the machine-readable equivalent; abort these steps.
7. If [parsing a floating date and time string](#) from the element's [datetime value](#) returns a [global date and time](#), that is the machine-readable equivalent; abort these steps.
8. If [parsing a week string](#) from the element's [datetime value](#) returns a [week](#), that is the machine-readable equivalent; abort these steps.
9. If the element's [datetime value](#) consists of only [ASCII digits](#), at least one of which is not U+0030 DIGIT ZERO (0), then the machine-readable equivalent is the base-ten interpretation of those digits, representing a year; abort these steps.
10. If [parsing a duration string](#) from the element's [datetime value](#) returns a [duration](#), that is the machine-readable equivalent; abort these steps.
11. There is no machine-readable equivalent.

NOTE:

The algorithms referenced above are intended to be designed such that for any arbitrary string *s*, only one of the algorithms returns a value. A more efficient approach might be to create a single algorithm that parses all these data types in one pass; developing such an algorithm is left as an exercise to the reader.

The **dateTime** IDL attribute must [reflect](#) the element's **datetime** content attribute.

EXAMPLE 269

The `<time>` element can be used to encode dates, for example in microformats. The following shows a hypothetical way of encoding an event using a variant on hCalendar that uses the `<time>` element:

```
<div class="vevent">
  <a class="url" href="https://www.web2con.com/">https://www.web2con.com/</a>
  <span class="summary">Web 2.0 Conference</span>:
  <time class="dtstart" datetime="2005-10-05">October 5</time> -
  <time class="dtend" datetime="2005-10-07">7</time>,
  at the <span class="location">Argent Hotel, San Francisco, CA</span>
</div>
```

EXAMPLE 270

Here, a fictional microdata vocabulary based on the Atom vocabulary is used with the [<time>](#) element to mark up a blog post's publication date.

```
<article vocab="https://n.example.org/" typeof="rfc4287">
  <h1 property="title">Big tasks</h1>
  <footer>Published <time property="published" datetime="2009-08-29">two days ago</time>.
</footer>
  <p property="content">Today, I went out and bought a bike for my kid.</p>
</article>
```

EXAMPLE 271

In this example, another article's publication date is marked up using [<time>](#), this time using the schema.org microdata vocabulary:

```
<article typeof="schema:BlogPosting">
  <h1 property="schema:headline">Small tasks</h1>
  <footer>Published <time property="schema:datePublished"
datetime="2009-08-30">yesterday</time>.</footer>
  <p property="schema:articleBody">I put a bike bell on his bike.</p>
</article>
```

EXAMPLE 272

In the following snippet, the [<time>](#) element is used to encode a date in the ISO8601 format, for later processing by a script:

```
<p>Our first date was <time datetime="2006-09-23">a Saturday</time>.</p>
```

In this second snippet, the value includes a time:

```
<p>We stopped talking at <time datetime="2006-09-24T05:00-07:00">5am the next
morning</time>.</p>
```

A script loaded by the page (and thus privy to the page's internal convention of marking up dates and times using the [<time>](#) element) could scan through the page and look at all the [<time>](#) elements therein to create an index of dates and times.

EXAMPLE 273

For example, this element conveys the string "Friday" with the additional semantic that the 18th of November 2011 is the meaning that corresponds to "Friday":

```
Today is <time datetime="2011-11-18">Friday</time>.
```

EXAMPLE 274

In this example, a specific time in the Pacific Standard Time timezone is specified:

```
Your next meeting is at <time datetime="2011-11-18T15:00-08:00">3pm</time>.
```

§ 4.5.17. The code element

Categories:

[Flow content.](#)

[Phrasing content.](#)

[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed [ARIA role attribute](#) values:

[Any role value.](#)

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

Uses [HTMLElement](#).

The `<code>` element [represents](#) a fragment of computer code. This could be an XML element name, a file name, a computer program, or any other string that a computer would recognize.

There is no formal way to indicate the language of computer code being marked up. Authors who wish to mark `<code>` elements with the language used, e.g., so that syntax highlighting scripts can use the right rules, can use the `class` attribute, e.g., by adding a class prefixed with "language-" to the element.

EXAMPLE 275

The following example shows how the element can be used in a paragraph to mark up element names and computer code, including punctuation.

```
<p>The <code>code</code> element represents a fragment of computer
code.</p>

<p>When you call the <code>activate()</code> method on the
<code>robotSnowman</code> object, the eyes glow.</p>

<p>The example below uses the <code>begin</code> keyword to indicate
the start of a statement block. It is paired with an <code>end</code>
keyword, which is followed by the <code>.</code> punctuation character
(full stop) to indicate the end of the program.</p>
```

EXAMPLE 276

The following example shows how a block of code could be marked up using the `pre` and `<code>` elements.

```
<pre><code class="language-pascal">var i: Integer;
begin
  i := 1;
end.</code></pre>
```

A class is used in that example to indicate the language used.

NOTE:

See the [<pre>](#) element for more details.

§ 4.5.18. The `var` element

Categories:

[Flow content.](#)
[Phrasing content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[Any role value.](#)

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)
Any aria-* attributes [applicable to the allowed roles.](#)

DOM interface:

Uses [HTMLElement](#).

The `<var>` element [represents](#) a variable. This could be an actual variable in a mathematical expression or programming context, an identifier representing a constant, a symbol identifying a physical quantity, a function parameter, or just be a term used as a placeholder in prose.

EXAMPLE 277

In the paragraph below, the letter "n" is being used as a variable in prose:

```
<p>If there are <var>n</var> pipes leading to the ice  
cream factory then I expect at <em>least</em> <var>n</var>  
flavors of ice cream to be available for purchase!</p>
```

For mathematics, in particular for anything beyond the simplest of expressions, MathML is more appropriate. However, the `<var>` element can still be used to refer to specific variables that are then mentioned in MathML expressions.

EXAMPLE 278

In this example, an equation is shown, with a legend that references the variables in the equation. The expression itself is marked up with MathML, but the variables are mentioned in the figure's legend using `<var>`.

```
<figure>
  <math>
    <mi>a</mi>
    <mo>=</mo>
    <msqrt>
      <msup><mi>b</mi><mn>2</mn></msup>
      <mi>+</mi>
      <msup><mi>c</mi><mn>2</mn></msup>
    </msqrt>
  </math>
  <figcaption>
    Using Pythagoras' theorem to solve for the hypotenuse <var>a</var> of
    a triangle with sides <var>b</var> and <var>c</var>
  </figcaption>
</figure>
```

EXAMPLE 279

Here, the equation describing mass-energy equivalence is used in a sentence, and the `<var>` element is used to mark the variables and constants in that equation:

```
<p>Then he turned to the blackboard and picked up the chalk. After a few moment's
thought, he wrote <var>E</var> = <var>m</var> <var>c</var><sup>2</sup>. The teacher
looked pleased.</p>
```

4.5.19. The samp element**Categories:**

[Flow content.](#)
[Phrasing content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[Any role value.](#)

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles.](#)

DOM interface:

Uses [HTMLElement](#).

The `<samp>` element [represents](#) sample or quoted output from another program or computing system.

NOTE:

See the `pre` and `<kbd>` elements for more details.

NOTE:

This element can be contrasted with the `<output>` element, which can be used to provide immediate output in a Web application.

EXAMPLE 280

This example shows the `<samp>` element being used inline:

```
<p>The computer said <samp>Too much cheese in tray
two</samp> but I didn't know what that meant.</p>
```

EXAMPLE 281

This second example shows a block of sample output. Nested `samp` and `<kbd>` elements allow for the styling of specific elements of the sample output using a style sheet. There's also a few parts of the `samp` that are annotated with even more detailed markup, to enable very precise styling. To achieve this, `<span>` elements are used.

```
<pre><samp><span class="prompt">jdoe@mowmow:~$</span> <kbd>ssh demo.example.com</kbd>
Last login: Tue Apr 12 09:10:17 2005 from mowmow.example.com on pts/1
Linux demo 2.6.10-grsec+gg3+e+fhs6b+nfs+gr0501+++p3+c4a+gr2b-reslog-v6.189 #1 SMP Tue Feb 1
11:22:36 PST 2005 i686 unknown

<span class="prompt">jdoe@demo:~$</span> <span class="cursor">_</span></samp></pre>
```

§ 4.5.20. The `kbd` element**Categories:**

[Flow content.](#)
[Phrasing content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[Any role value.](#)

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any `aria-*` attributes [applicable to the allowed roles](#).

DOM interface:

Uses [HTMLElement](#).

The `<kbd>` element [represents](#) user input (typically keyboard input, although it may also be used to represent other input, such as voice commands).

When the `<kbd>` element is nested inside a `<samp>` element, it represents the input as it was echoed by the system.

When the `<kbd>` element *contains* a `<samp>` element, it represents input based on system output, for example invoking a menu item.

When the `<kbd>` element is nested inside another `<kbd>` element, it represents an actual key or other single unit of input as appropriate for the input mechanism.

EXAMPLE 282

Here the `<kbd>` element is used to indicate keys to press:

```
<p>To make George eat an apple, press <kbd><kbd>Shift</kbd>+<kbd>F3</kbd></kbd></p>
```

In this second example, the user is told to pick a particular menu item. The outer `<kbd>` element marks up a block of input, with the inner `<kbd>` elements representing each individual step of the input, and the `<samp>` elements inside them indicating that the steps are input based on something being displayed by the system, in this case menu labels:

```
<p>To make George eat an apple, select
  <kbd><kbd><samp>File</samp></kbd>|<kbd><samp>Eat Apple...</samp></kbd></kbd>
</p>
```

Such precision isn't necessary; the following is equally fine:

```
<p>To make George eat an apple, select <kbd>File | Eat Apple...</kbd></p>
```

4.5.21. The sub and sup elements

Categories:

[Flow content.](#)
[Phrasing content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[Any role value.](#)

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

Use `HTMLElement`.

The `<sup>` element [represents](#) a superscript and the sub element [represents](#) a subscript.

These elements must be used only to mark up typographical conventions with specific meanings, not for typographical presentation for presentation's sake. For example, it would be inappropriate for the sub and `<sup>` elements to be used in the name of the LaTeX document preparation system. In general, authors should use these elements only if the *absence* of those elements would change the meaning of the content.

In certain languages, superscripts are part of the typographical conventions for some abbreviations.

EXAMPLE 283

```
<p>The most beautiful women are
<span lang="fr"><abbr>M<sup>lle</sup></abbr> Gwendoline</span> and
<span lang="fr"><abbr>M<sup>me</sup></abbr> Denise</span>.</p>
```

The `<sub>` element can be used inside a `<var>` element, for variables that have subscripts.

EXAMPLE 284

Here, the `<sub>` element is used to represent the subscript that identifies the variable in a family of variables:

```
<p>The coordinate of the <var>i</var>th point is
(<var>x<sub><var>i</var></sub></var>, <var>y<sub><var>i</var></sub></var>).
For example, the 10th point has coordinate
(<var>x<sub>10</sub></var>, <var>y<sub>10</sub></var>).</p>
```

Mathematical expressions often use subscripts and superscripts. Authors are encouraged to use MathML for marking up mathematics, but authors may opt to use `sub` and `sup` if detailed mathematical markup is not desired. [\[MATHML\]](#)

EXAMPLE 285

```
<var>E</var>=<var>m</var><var>c</var><sup>2</sup>

f(<var>x</var>, <var>n</var>) = log<sub>4</sub><var>x</var><sup><var>n</var></sup>
```

§ 4.5.22. The `i` element**Categories:**

[Flow content.](#)
[Phrasing content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[Any role value.](#)

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)
 Any aria-* attributes [applicable to the allowed roles.](#)

DOM interface:

Uses [HTMLElement](#).

The `<i>` element [represents](#) a span of text in an alternate voice or mood, or otherwise offset from the normal prose in a manner indicating a different quality of text, such as a taxonomic designation, a technical term, an idiomatic phrase from another language, transliteration, a thought, or a ship name in Western texts.

Terms in languages different from the main text should be annotated with [lang](#) attributes (or, in XML, [lang](#) attributes in the [XML namespace](#)).

EXAMPLE 286

The examples below show uses of the [i](#) element:

```
<p>The <i class="taxonomy">Felis silvestris catus</i> is cute.</p>
<p>The term <i>prose content</i> is defined above.</p>
<p>There is a certain <i lang="fr">je ne sais quoi</i> in the air.</p>
```

In the following example, a dream sequence is marked up using [i](#) elements.

```
<p>Raymond tried to sleep.</p>
<p><i>The ship sailed away on Thursday</i>, he
dreamt. <i>The ship had many people aboard, including a beautiful
princess called Carey. He watched her, day-in, day-out, hoping she
would notice him, but she never did.</i></p>
<p><i>Finally one night he picked up the courage to speak with
her—</i></p>
<p>Raymond woke with a start as the fire alarm rang out.</p>
```

Authors can use the `class` attribute on the `i` element to identify why the element is being used, so that if the style of a particular use (e.g., dream sequences as opposed to taxonomic terms) is to be changed at a later date, the author doesn't have to go through the entire document (or series of related documents) annotating each use.

Authors are encouraged to consider whether other elements might be more applicable than the [i](#) element, for instance the [em](#) element for marking up stress emphasis, or the [dfn](#) element to mark up the defining instance of a term.

NOTE:

Style sheets can be used to format [i](#) elements, just like any other element can be restyled. Thus, it is not the case that content in [i](#) elements will necessarily be italicized.

4.5.23. The `b` element

Categories:

[Flow content.](#)
[Phrasing content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[Any role value.](#)

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

Uses [HTMLElement](#).

The `<b>` element [represents](#) a span of text to which attention is being drawn for utilitarian purposes without conveying any extra importance and with no implication of an alternate voice or mood, such as key words in a document abstract, product names in a review, actionable words in interactive text-driven software, or an article lede.

EXAMPLE 287

The following example shows a use of the `<b>` element to highlight key words without marking them up as important:

```
<p>The <b>frobonitor</b> and <b>barbinator</b> components are fried.</p>
```

EXAMPLE 288

In the following example, objects in a text adventure are highlighted as being special by use of the `<b>` element.

```
<p>You enter a small room. Your <b>sword</b> glows  
brighter. A <b>rat</b> scurries past the corner wall.</p>
```

EXAMPLE 289

Another case where the `<b>` element is appropriate is in marking up the lede (or lead) sentence or paragraph. The following example shows how a [BBC article about kittens adopting a rabbit as their own](#) could be marked up:

```
<article>  
  <h2>Kittens 'adopted' by pet rabbit</h2>  
  <p><b class="lede">Six abandoned kittens have found an  
unexpected new mother figure – a pet rabbit.</b></p>  
  <p>Veterinary nurse Melanie Humble took the three-week-old  
kittens to her Aberdeen home.</p>  
[...]
```

As with the `<i>` element, authors can use the `class` attribute on the `<b>` element to identify why the element is being used, so that if the style of a particular use is to be changed at a later date, the author doesn't have to go through annotating each use.

The `<b>` element should be used as a last resort when no other element is more appropriate. In particular, headings should use the `h1` to `h6` elements, stress emphasis should use the `<em>` element, importance should be denoted with the `<strong>` element, and text marked or highlighted should use the `mark` element.

EXAMPLE 290

The following would be *incorrect* usage:

```
<p><b>WARNING!</b> Do not frob the barbinator!</p>
```

In the previous example, the correct element to use would have been `<strong>`, not `<b>`.

NOTE:

Style sheets can be used to format `<b>` elements, just like any other element can be restyled. Thus, it is not the case that content in `<b>` elements will necessarily be boldened.

§ 4.5.24. The `u` element

Categories:

[Flow content.](#)
[Phrasing content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[Any role value.](#)

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles.](#)

DOM interface:

Uses [HTMLElement](#).

The `<u>` element [represents](#) a span of text with an unarticulated, though explicitly rendered, non-textual annotation, such as labeling the text as being a proper name in Chinese text (a Chinese proper name mark), or labeling the text as being misspelt.

In most cases, another element is likely to be more appropriate: for marking stress emphasis, the `<em>` element should be used; for marking key words or phrases either the `<b>` element or the `<mark>` element should be used, depending on the context; for marking book titles, the `<cite>` element should be used; for labeling text with explicit textual annotations, the `<ruby>` element should be used; for technical terms, taxonomic designation, transliteration, a thought, or for labeling ship names in Western texts, the `<i>` element should be used.

NOTE:

The default rendering of the `<u>` element in visual presentations clashes with the conventional rendering of hyperlinks (underlining). Authors are encouraged to avoid using the `<u>` element where it could be confused for a hyperlink.

4.5.25. The mark element

Categories:

[Flow content.](#)

[Phrasing content.](#)

[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[Any role value.](#)

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles.](#)

DOM interface:

Uses [HTMLElement](#).

The `<mark>` element [represents](#) a run of text in one document marked or highlighted for reference purposes, due to its relevance in another context. When used in a quotation or other block of text referred to from the prose, it indicates a highlight

that was not originally present but which has been added to bring the reader's attention to a part of the text that might not have been considered important by the original author when the block was originally written, but which is now under previously unexpected scrutiny. When used in the main prose of a document, it indicates a part of the document that has been highlighted due to its likely relevance to the user's current activity.

EXAMPLE 291

This example shows how the `<mark>` element can be used to bring attention to a particular part of a quotation:

```
<p lang="en-US">Consider the following quote:</p>
<blockquote lang="en-GB">
  <p>Look around and you will find, no-one's really
    <mark>colour</mark> blind.</p>
</blockquote>
<p lang="en-US">As we can tell from the <em>spelling</em> of the word,
  the person writing this quote is clearly not American.</p>
```

(If the goal was to mark the element as misspelt, however, the `<u>` element, possibly with a class, would be more appropriate.)

EXAMPLE 292

Another example of the `<mark>` element is highlighting parts of a document that are matching some search string. If someone looked at a document, and the server knew that the user was searching for the word "kitten", then the server might return the document with one paragraph modified as follows:

```
<p>I also have some <mark>kitten</mark>s who are visiting me
these days. They're really cute. I think they like my garden! Maybe I
should adopt a <mark>kitten</mark>.</p>
```

EXAMPLE 293

In the following snippet, a paragraph of text refers to a specific part of a code fragment.

```
<p>The highlighted part below is where the error lies:</p>
<pre><code>var i: Integer;
begin
  i := <mark>1.1</mark>;
end.</code></pre>
```

This is separate from *syntax highlighting*, for which `span` is more appropriate. Combining both, one would get:

```
<p>The highlighted part below is where the error lies:</p>
<pre><code><span class=keyword>var</span> <span class=ident>i</span>: <span
class=type>Integer</span>;
<span class=keyword>begin</span>
  <span class=ident>i</span> := <span class=literal><mark>1.1</mark></span>;
<span class=keyword>end</span>.</code></pre>
```

EXAMPLE 294

This is another example showing the use of `mark` to highlight a part of quoted text that was originally not emphasized. In this example, common typographic conventions have led the author to explicitly style `<mark>` elements in quotes to render in italics.

```
<head>
  <style>
    blockquote mark, q mark {
      font: inherit; font-style: italic;
      text-decoration: none;
      background: transparent; color: inherit;
    }
    .bubble em {
      font: inherit; font-size: larger;
      text-decoration: underline;
    }
  </style>
</head>
<article>
  <h1>She knew</h1>
  <p>Did you notice the subtle joke in the joke on panel 4?</p>
  <blockquote>
    <p class="bubble">I didn't <em>want</em> to believe. <mark>Of course
    on some level I realized it was a known-plaintext attack.</mark> But I
    couldn't admit it until I saw for myself.</p>
  </blockquote>
  <p>(Emphasis mine.) I thought that was great. It's so pedantic, yet it
  explains everything neatly.</p>
</article>
```

NOTE:

Note, incidentally, the distinction between the `<em>` element in this example, which is part of the original text being quoted, and the `<mark>` element, which is highlighting a part for comment.

EXAMPLE 295

The following example shows the difference between denoting the *importance* of a span of text (`strong`) as opposed to denoting the *relevance* of a span of text (`mark`). It is an extract from a textbook, where the extract has had the parts relevant to the exam highlighted. The safety warnings, important though they may be, are apparently not relevant to the exam.

```
<h3>Wormhole Physics Introduction</h3>

<p><mark>A wormhole in normal conditions can be held open for a
maximum of just under 39 minutes.</mark> Conditions that can increase
the time include a powerful energy source coupled to one or both of
the gates connecting the wormhole, and a large gravity well (such as a
black hole).</p>

<p><mark>Momentum is preserved across the wormhole. Electromagnetic
radiation can travel in both directions through a wormhole,
but matter cannot.</mark></p>

<p>When a wormhole is created, a vortex normally forms.
<strong>Warning: The vortex caused by the wormhole opening will
annihilate anything in its path.</strong> Vortexes can be avoided when
using sufficiently advanced dialing technology.</p>

<p><mark>An obstruction in a gate will prevent it from accepting a
wormhole connection.</mark></p>
```

§ 4.5.26. The `bdi` element

Categories:

[Flow content.](#)
[Phrasing content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Also, the `dir` global attribute has special semantics on this element.

Allowed [ARIA role attribute](#) values:

[Any role value.](#)

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any `aria-*` attributes [applicable to the allowed roles](#).

DOM interface:

Uses [HTMLElement](#).

The `<bdi>` element [represents](#) a span of text that is to be isolated from its surroundings for the purposes of bidirectional text formatting. [\[BIDI\]](#)

NOTE:

The `dir` global attribute defaults to `auto` on this element (it never inherits from the parent element like with other elements).

NOTE:

This element [has rendering requirements involving the bidirectional algorithm](#).

EXAMPLE 296

This element is especially useful when embedding user-generated content with an unknown directionality.

In this example, usernames are shown along with the number of posts that the user has submitted. If the `<bdi>` element were not used, the username of the Arabic user would end up confusing the text (the bidirectional algorithm would put the colon and the number "3" next to the word "User" rather than next to the word "posts").

```
<ul>
  <li>User <bdi>jcranmer</bdi>: 12 posts.
  <li>User <bdi>hober</bdi>: 5 posts.
  <li>User <bdi>إيان</bdi>: 3 posts.
</ul>
```

- User jcranmer: 12 posts.
- User hober: 5 posts.
- User إيان: 3 posts.

Figure 1 When using the `<bdi>` element, the username acts as expected.

- User **jcranmer**: 12 posts.
- User **hober**: 5 posts.
- User 3 إيان: posts.

Figure 2 If the `<bdi>` element were to be replaced by a `<b>` element, the username would confuse the bidirectional algorithm and the third bullet would end up saying "User 3 :", followed by the Arabic name (right-to-left), followed by "posts" and a period.

4.5.27. The `bdo` element

Categories:

[Flow content.](#)
[Phrasing content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Also, the `dir` global attribute has special semantics on this element.

Allowed [ARIA role attribute](#) values:

[Any role value.](#)

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any `aria-*` attributes [applicable to the allowed roles](#).

DOM interface:

Uses [HTMLElement](#).

The `<bdo>` element [represents](#) explicit text directionality formatting control for its children. It allows authors to override the Unicode bidirectional algorithm by explicitly specifying a direction override. [\[BIDI\]](#)

Authors must specify the [dir](#) attribute on this element, with the value [ltr](#) to specify a left-to-right override and with the value [rtl](#) to specify a right-to-left override. The [auto](#) value must not be specified.

NOTE:

This element [has rendering requirements involving the bidirectional algorithm](#).

§ 4.5.28. The span element

Categories:

[Flow content](#).
[Phrasing content](#).
[Palpable content](#).

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content](#).

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[Any role value](#).

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLSpanElement : HTMLElement {};
```

The `<span>` element doesn't mean anything on its own, but can be useful when used together with the [Global attributes](#), e.g., [class](#), [lang](#), or [dir](#). It [represents](#) its children.

EXAMPLE 297

In this example, a code fragment is marked up using `<span>` elements and `class` attributes so that its keywords and identifiers can be color-coded from CSS:

```
<pre><code class="lang-c"><span class="keyword">for</span> (<span class="ident">j</span> = 0;
<span class="ident">j</span> <span class="ident">&lt; 256; <span class="ident">j</span>++) {
  <span class="ident">i_t3</span> = (<span class="ident">i_t3</span> & 0x1ffff) | (<span
class="ident">j</span> &lt;&lt; 17);
  <span class="ident">i_t6</span> = ((((((<span class="ident">i_t3</span> >> 3) ^ <span
class="ident">i_t3</span>) >> 1) ^ <span class="ident">i_t3</span>) >> 8) ^ <span
class="ident">i_t3</span>) >> 5) & 0xff;
  <span class="keyword">if</span> (<span class="ident">i_t6</span> == <span
class="ident">i_t1</span>)
    <span class="keyword">break</span>;
}</code></pre>
```

§ 4.5.29. The br element

Categories:

[Flow content](#).

[Phrasing content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Nothing.](#)

Tag omission in text/html:

No [end tag](#)

Content attributes:

[Global attributes](#)

Allowed [ARIA role attribute](#) values:

[Any role value.](#)

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any [aria-* attributes](#) [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLBRElement : HTMLElement {};
```

The `<br>` element [represents](#) a line break.

NOTE:

While line breaks are usually represented in visual media by physically moving subsequent text to a new line, a style sheet or user agent would be equally justified in causing line breaks to be rendered in a different manner, for instance as green dots, or as extra spacing.

`<br>` elements must be used only for line breaks that are actually part of the content, as in poems or addresses.

EXAMPLE 298

The following example is correct usage of the `<br>` element:

```
<p>P. Sherman<br>
42 Wallaby Way<br>
Sydney</p>
```

`<br>` elements must not be used for separating thematic groups in a paragraph.

EXAMPLE 299

The following examples are non-conforming, as they abuse the `<br>` element:

```
<p><a ...>34 comments.</a><br>
<a ...>Add a comment.</a></p>

<p><label>Name: <input name="name"></label><br>
<label>Address: <input name="address"></label></p>
```

Here are alternatives to the above, which are correct:

```
<p><a ...>34 comments.</a></p>
<p><a ...>Add a comment.</a></p>

<p><label>Name: <input name="name"></label></p>
<p><label>Address: <input name="address"></label></p>
```

If a [paragraph](#) consists of nothing but a single `<br>` element, it represents a placeholder blank line (e.g., as in a template). Such blank lines must not be used for presentation purposes.

Any content inside `<br>` elements must not be considered part of the surrounding text.

NOTE:

This element [has rendering requirements involving the bidirectional algorithm](#).

4.5.30. The `wbr` element

Categories:

[Flow content](#).

[Phrasing content](#).

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Nothing](#).

Tag omission in text/html:

No [end tag](#)

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[Any role value](#).

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

Uses [HTMLElement](#).

The `<wbr>` element [represents](#) a line break opportunity.

EXAMPLE 300

In the following example, someone is quoted as saying something which, for effect, is written as one long word. However, to ensure that the text can be wrapped in a readable fashion, the individual words in the quote are separated using a `<wbr>` element.

```
<p>So then he pointed at the tiger and screamed  
"there<wbr>is<wbr>no<wbr>way<wbr>you<wbr>are<wbr>ever<wbr>going<wbr>to<wbr>catch<wbr>me"!</p>
```

EXAMPLE 301

Here, especially long lines of code in a program listing have suggested wrapping points given using `<wbr>` elements.

```
<pre>...  
Helm heading = Helm.HeadingFactory(HeadingCoordinates[1], <wbr>HeadingCoordinates[2],  
<wbr>HeadingCoordinates[3], <wbr>HeadingCoordinates[4]);  
Course course = Helm.CourseFactory(Heading, <wbr>Maps.MapFactoryFromHeading(heading),  
<wbr>Speeds.GetMaximumSpeed().ConvertToWarp());  
...</pre>
```

Any content inside `<wbr>` elements must not be considered part of the surrounding text.

EXAMPLE 302

```
var wbr = document.createElement("wbr");wbr.textContent = "This is wrong";
document.body.appendChild(wbr);
```

NOTE:

This element [has rendering requirements involving the bidirectional algorithm](#).

§ 4.5.31. Usage summary

This section is non-normative.

Element	Purpose	Example
<a>	Hyperlinks	EXAMPLE 303 Visit my drinks page.
	Stress emphasis	EXAMPLE 304 I must say I adore lemonade.
	Importance	EXAMPLE 305 This tea is very hot .
<small>	Side comments	EXAMPLE 306 These grapes are made into wine. Alcohol is addictive.
<s>	Inaccurate text	EXAMPLE 307 Price: ~~£4.50~~ £2.00!
<cite>	Titles of works	EXAMPLE 308 The case <i>Hugo v. Danielle</i> is relevant here.
<q>	Quotations	EXAMPLE 309 The judge said <i>You can drink water from the fish tank</i> but advised against it.
<dfn>	Defining instance	EXAMPLE 310 The term <i>organic food</i> refers to food produced without synthetic chemicals.
<abbr>	Abbreviations	EXAMPLE 311 Organic food in Ireland is certified by the <i>Irish Organic Farmers and Growers Association</i> .
<ruby> , <rb> , <rp> , <rt> , rtc	Ruby annotations	EXAMPLE 312 <code><ruby> <rb>OJ <rp>(<rtc><rt>Orange Juice</rtc><rp>)</ruby></code>
<data>	Machine-readable equivalent	EXAMPLE 313 Available starting today! <code><data value="UPC:022014640201">North Coast Organic Apple Cider</data></code>
<time>	Machine-readable equivalent of date- or time-related data	EXAMPLE 314 Available starting on <code><time datetime="2011-11-18">November 18th</time></code> !
<code>	Computer code	EXAMPLE 315 The <code>fruitdb</code> program can be used for tracking fruit production.

Element	Purpose	Example
<var>	Variables	EXAMPLE 316 If there are <code><var>n</var></code> fruit in the bowl, at least <code><var>n</var>+2</code> will be ripe.
<samp>	Computer output	EXAMPLE 317 The computer said <code><samp>Unknown error -3</samp></code>.
<kbd>	User input	EXAMPLE 318 Hit <code><kbd>F1</kbd></code> to continue.
<sub>	Subscripts	EXAMPLE 319 Water is H<code><sub>2</sub></code>O.
<sup>	Superscripts	EXAMPLE 320 The Hydrogen in heavy water is usually <code><sup>2</sup></code>H.
<i>	Alternative voice	EXAMPLE 321 Lemonade consists primarily of <code><i>Citrus limon</i></code>.
	Keywords	EXAMPLE 322 Take a <code>lemon</code> and squeeze it with a <code>juicer</code>.
<u>	Annotations	EXAMPLE 323 The mixture of apple juice and <code><u class="spelling">elderflower</u></code> juice is very pleasant.
<mark>	Highlight	EXAMPLE 324 Elderflower cordial, with one <code><mark>part</mark></code> cordial to ten <code><mark>part</mark></code>s water, stands a <code><mark>part</mark></code> from the rest.
<bdi>	Text directionality isolation	EXAMPLE 325 The recommended restaurant is <code><bdi lang="">My Juice Café (At The Beach)</bdi></code>.
<bdo>	Text directionality formatting	EXAMPLE 326 The proposal is to write English, but in reverse order. "Juice" would become "<code><bdo dir=rtl>Juice</bdo></code>"
	Other	EXAMPLE 327 In French we call it <code>sirop de sureau</code>.

	Line break	EXAMPLE 328 Simply Orange Juice Company<code>
</code>Apopka, FL 32703<code>
</code>U.S.A.
<wbr>	Line breaking opportunity	EXAMPLE 329 www.simply<code><wbr></code>orange<code><wbr></code>juice.com

§ 4.6. Edits

The [<ins>](#) and [<del>](#) elements represent edits to the document.

§ 4.6.1. The ins element

Categories:

- [Flow content.](#)
- [Phrasing content.](#)
- [Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Transparent](#).

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

`cite` - Link to the source of the quotation or more information about the edit

`datetime` - Date and (optionally) time of the change

Allowed [ARIA role attribute](#) values:

[Any role value](#).

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any `aria-*` attributes [applicable to the allowed roles](#).

DOM interface:

Uses the `HTMLModElement` interface.

The `<ins>` element [represents](#) an addition to the document.

EXAMPLE 330

The following represents the addition of a single paragraph:

```
<aside>
  <ins>
    <p> I like fruit. </p>
  </ins>
</aside>
```

As does the following, because everything in the `<aside>` element here counts as [phrasing content](#) and therefore there is just one [paragraph](#):

```
<aside>
  <ins>
    Apples are <em>tasty</em>.
  </ins>
  <ins>
    So are pears.
  </ins>
</aside>
```

`<ins>` elements should not cross [implied paragraph](#) boundaries.

EXAMPLE 331

The following example represents the addition of two paragraphs, the second of which was inserted in two parts. The first `<ins>` element in this example thus crosses a paragraph boundary, which is considered poor form.

```
<aside> <!-- don't do this -->
  <ins datetime="2005-03-16 00:00Z">
    <p> I like fruit. </p>
    Apples are <em>tasty</em>.
  </ins>
  <ins datetime="2007-12-19 00:00Z">
    So are pears.
  </ins>
</aside>
```

Here is a better way of marking this up. It uses more elements, but none of the elements cross implied paragraph boundaries.

```
<aside>
  <ins datetime="2005-03-16 00:00Z">
    <p> I like fruit. </p>
  </ins>
  <ins datetime="2005-03-16 00:00Z">
    Apples are <em>tasty</em>.
  </ins>
  <ins datetime="2007-12-19 00:00Z">
    So are pears.
  </ins>
</aside>
```

§ **4.6.2. The `del` element****Categories:**

[Flow content.](#)
[Phrasing content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Transparent.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

`cite` - Link to the source of the quotation or more information about the edit

`datetime` - Date and (optionally) time of the change

Allowed [ARIA role attribute](#) values:

[Any role value.](#)

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

Uses the `HTMLModElement` interface.

The `<del>` element [represents](#) a removal from the document.

`<del>` elements should not cross [implied paragraph](#) boundaries.

EXAMPLE 332

The following shows a "to do" list where items that have been done are crossed-off with the date and time of their completion.

```
<h1>To Do</h1>
<ul>
  <li>Empty the dishwasher</li>
  <li><del datetime="2009-10-11T01:25-07:00">Watch Walter Lewin's lectures</del></li>
  <li><del datetime="2009-10-10T23:38-07:00">Download more tracks</del></li>
  <li>Buy a printer</li>
</ul>
```

§ 4.6.3. Attributes common to `ins` and `<del>` elements

The **cite** attribute may be used to specify the address of a document that explains the change. When that document is long, for instance the minutes of a meeting, authors are encouraged to include a fragment identifier pointing to the specific part of that document that discusses the change.

If the **cite** attribute is present, it must be a [valid URL potentially surrounded by spaces](#) that explains the change. To obtain the corresponding citation link, the value of the attribute must be [resolved](#) relative to the element. User agents may allow users to follow such citation links, but they are primarily intended for private use (e.g., by server-side scripts collecting statistics about a site's edits), not for readers.

The **datetime** attribute may be used to specify the time and date of the change.

If present, the **datetime** attribute's value must be a [valid date string with optional time](#).

User agents must parse the **datetime** attribute according to the [parse a date or time string](#) algorithm. If that doesn't return a [date](#) or a [global date and time](#), then the modification has no associated timestamp (the value is non-conforming; it is not a [valid date string with optional time](#)). Otherwise, the modification is marked as having been made at the given [date](#) or [global date and time](#). If the given value is a [global date and time](#) then user agents should use the associated time-zone offset information to determine which time zone to present the given datetime in.

This value may be shown to the user, but it is primarily intended for private use.

The `<ins>` and `<del>` elements must implement the [HTMLModElement](#) interface:

```
interface HTMLModElement : HTMLElement {
  attribute DOMString cite;
  attribute DOMString dateTime;
};
```

The **cite** IDL attribute must [reflect](#) the element's [cite](#) content attribute. The **dateTime** IDL attribute must [reflect](#) the element's [datetime](#) content attribute.

§ 4.6.4. Edits and paragraphs

This section is non-normative.

Since the `ins` and `<del>` elements do not affect [paragraphing](#), it is possible, in some cases where paragraphs are [implied](#) (without explicit `<p>` elements), for an `ins` or `<del>` element to span both an entire paragraph or other non-[phrasing content](#) elements and part of another paragraph. For example:

```

<section>
  <ins>
    <p>
      This is a paragraph that was inserted.
    </p>
    This is another paragraph whose first sentence was inserted
    at the same time as the paragraph above.
  </ins>
  This is a second sentence, which was there all along.
</section>

```

By only wrapping some paragraphs in `<p>` elements, one can even get the end of one paragraph, a whole second paragraph, and the start of a third paragraph to be covered by the same `ins` or `<del>` element (though this is very confusing, and not considered good practice):

```

<section> This is the first paragraph. <ins>This sentence was
  inserted.
  <p>This second paragraph was inserted.</p>
  This sentence was inserted too.</ins> This is the
  third paragraph in this example.
  <!-- (don't do this) -->
</section>

```

However, due to the way [implied paragraphs](#) are defined, it is not possible to mark up the end of one paragraph and the start of the very next one using the same `ins` or `<del>` element. You instead have to use one (or two) `p` element(s) and two `ins` or `<del>` elements, as for example:

```

<section>
  <p>This is the first paragraph. <del>This sentence was
  deleted.</del></p>
  <p><del>This sentence was deleted too.</del> That
  sentence needed a separate &lt;del> element.</p>
</section>

```

Partly because of the confusion described above, authors are strongly encouraged to always mark up all paragraphs with the `<p>` element, instead of having `ins` or `<del>` elements that cross [implied paragraphs](#) boundaries.

§ 4.6.5. Edits and lists

This section is non-normative.

The content models of the `ol` and `<ul>` elements do not allow `ins` and `<del>` elements as children. Lists always represent all their items, including items that would otherwise have been marked as deleted.

To indicate that an item is inserted or deleted, an `ins` or `del` element can be wrapped around the contents of the `<li>` element. To indicate that an item has been replaced by another, a single `<li>` element can have one or more `<del>` elements followed by one or more `<ins>` elements.

EXAMPLE 333

In the following example, a list that started empty had items added and removed from it over time. The bits in the example that have been emphasized show the parts that are the "current" state of the list. The list item numbers don't take into account the edits, though.

```
<h1>Stop-ship bugs</h1>
<ol>
  <li><ins datetime="2008-02-12T15:20Z">Bug 225:
    Rain detector doesn't work in snow</ins></li>
  <li><del datetime="2008-03-01T20:22Z"><ins datetime="2008-02-14T12:02Z">Bug 228:
    Water buffer overflows in April</ins></del></li>
  <li><ins datetime="2008-02-16T13:50Z">Bug 230:
    Water heater doesn't use renewable fuels</ins></li>
  <li><del datetime="2008-02-20T21:15Z"><ins datetime="2008-02-16T14:25Z">Bug 232:
    Carbon dioxide emissions detected after startup</ins></del></li>
</ol>
```

EXAMPLE 334

In the following example, a list that started with just fruit was replaced by a list with just colors.

```
<h1>List of <del>fruits</del><ins>colors</ins></h1>
<ul>
  <li><del>Lime</del><ins>Green</ins></li>
  <li><del>Apple</del></li>
  <li>Orange</li>
  <li><del>Pear</del></li>
  <li><ins>Teal</ins></li>
  <li><del>Lemon</del><ins>Yellow</ins></li>
  <li>Olive</li>
  <li><ins>Purple</ins></li>
</ul>
```

§ 4.6.6. Edits and tables

This section is non-normative.

The elements that form part of the table model have complicated content model requirements that do not allow for the `ins` and `del` elements, so indicating edits to a table can be difficult.

To indicate that an entire row or an entire column has been added or removed, the entire contents of each cell in that row or column can be wrapped in `ins` or `del` elements (respectively).

EXAMPLE 335

Here, a table's row has been added:

```
<table>
  <thead>
    <tr> <th> Game name          <th> Game publisher  <th> Verdict
  <tbody>
    <tr> <td> Diablo 2            <td> Blizzard    <td> 8/10
    <tr> <td> Portal              <td> Valve        <td> 10/10
    <tr> <td> <ins>Portal 2</ins> <td> <ins>Valve</ins> <td> <ins>10/10</ins>
  </tbody>
</table>
```

Here, a column has been removed (the time at which it was removed is given also, as is a link to the page explaining why):

```
<table>
  <thead>
    <tr> <th> Game name          <th> Game publisher  <th> <del cite="/<ins>Verdict</del>
    <td> <del cite="/<ins>8/10</del>
    <td> <del cite="/<ins>10/10</del>
    <td> <del cite="/<ins>10/10</del>
  </tbody>
</table>
```

Generally speaking, there is no good way to indicate more complicated edits (e.g., that a cell was removed, moving all subsequent cells up or to the left).

§ 4.7. Embedded content

§ 4.7.1. Introduction

This section is non-normative.

To embed an image in HTML, when there is only a single image resource, use the `<img>` element and with its `src` and `alt` attributes.

EXAMPLE 336

```
<h2>From today's featured article</h2>

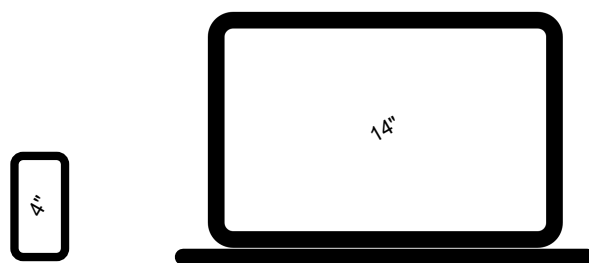
<p><b><a href="/wiki/Marie_Lloyd">Marie Lloyd</a></b> (1870–1922)
was an English <a href="/wiki/Music_hall">music hall</a> singer, ...
```

However, there are a number of situations for which the author might wish to use multiple image resources that the user agent can choose from:

- Different users might have different environmental characteristics:
 - The users' physical screen size might be different from one another.

EXAMPLE 337

A mobile phone's screen might be 4 inches diagonally, while a laptop's screen might be 14 inches diagonally.

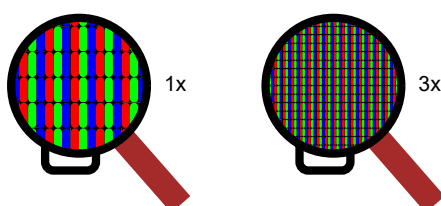
**NOTE:**

This is only relevant when an image's rendered size depends on the [viewport](#) size.

- The users' screen pixel density might be different from one another.

EXAMPLE 338

A mobile phone's screen might have three times as many physical pixels per inch compared to another mobile phone's screen, regardless of their physical screen size.



- The users' zoom level might be different from one another, or might change for a single user over time.

EXAMPLE 339

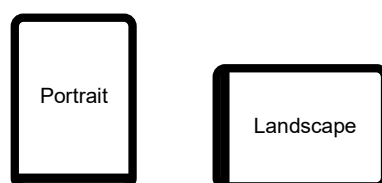
A user might zoom in to a particular image to be able to get a more detailed look.

The zoom level and the screen pixel density (the previous point) can both affect the number of physical screen pixels per CSS pixel. This ratio is usually referred to as **device-pixel-ratio**.

- The users' screen orientation might be different from one another, or might change for a single user over time.

EXAMPLE 340

A tablet can be held upright or rotated 90 degrees, so that the screen is either "portrait" or "landscape".



- The users' network speed, network latency and bandwidth cost might be different from one another, or might change for a single user over time.

EXAMPLE 341

A user might be on a fast, low-latency and constant-cost connection while at work, on a slow, low-latency and constant-cost connection while at home, and on a variable-speed, high-latency and variable-cost connection anywhere else.

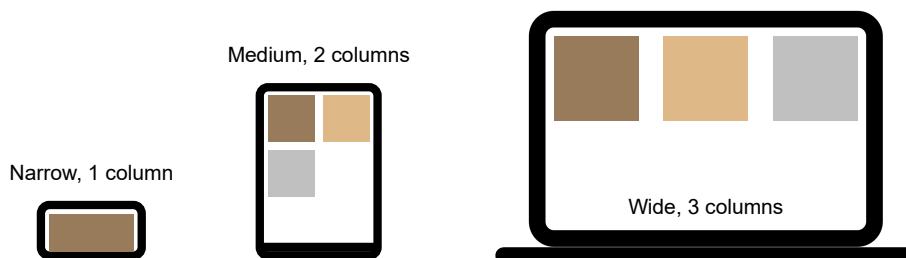
- Authors might want to show the same image content but with different rendered size depending on, usually, the width of the [viewport](#). This is usually referred to as **viewport-based selection**.

EXAMPLE 342

A Web page might have a banner at the top that always spans the entire [viewport](#) width. In this case, the rendered size of the image depends on the physical size of the screen (assuming a maximised browser window).

**EXAMPLE 343**

Another Web page might have images in columns, with a single column for screens with a small physical size, two columns for screens with medium physical size, and three columns for screens with big physical size, with the images varying in rendered size in each case to fill up the [viewport](#). In this case, the rendered size of an image might be *bigger* in the one-column layout compared to the two-column layout, despite the screen being smaller.



- Authors might want to show different image content depending on the rendered size of the image. This is usually referred to as **art direction**.

EXAMPLE 344

When a Web page is viewed on a screen with a large physical size (assuming a maximised browser window), the author might wish to include some less relevant parts surrounding the critical part of the image. When the same Web page is viewed on a screen with a small physical size, the author might wish to show only the critical part of the image.



- Authors might want to show the same image content but using different image formats, depending on which image formats the user agent supports. This is usually referred to as **image format-based selection**.

EXAMPLE 345

A Web page might have some images in the JPEG, WebP and JPEG XR image formats, with the latter two having better compression abilities compared to JPEG. Since different user agents can support different image formats, with some formats offering better compression ratios, the author would like to serve the better formats to user agents that support them, while providing JPEG fallback for user agents that don't.

The above situations are not mutually exclusive. For example, it is reasonable to combine different resources for different [device-pixel-ratio](#) with different resources for [art direction](#).

While it is possible to solve these problems using scripting, doing so introduces some other problems:

- Some user agents aggressively download images specified in the HTML markup, before scripts have had a chance to run, so that Web pages complete loading sooner. If a script changes which image to download, the user agent will potentially start two separate downloads, which can instead cause worse page loading performance.
- If the author avoids specifying any image in the HTML markup and instead instantiates a single download from script, that avoids the double download problem above but instead it makes no image be downloaded at all for users with scripting disabled and it disables the aggressive image downloading optimization.

With this in mind, this specification introduces a number of features to address the above problems in a declarative manner.

Device-pixel-ratio-based selection when the rendered size of the image is fixed

The [src](#) and [srcset](#) attributes on the [](#) element can be used, using the x descriptor, to provide multiple images that only vary in their size (the smaller image is a scaled-down version of the bigger image).

NOTE:

The x descriptor is not appropriate when the rendered size of the image depends on the [viewport](#) width ([viewport-based selection](#)), but can be used together with [art direction](#).

EXAMPLE 346

```
<h2>From today's featured article</h2>

<p><b><a href="/wiki/Marie_Lloyd">Marie Lloyd</a></b> (1870–1922)
was an English <a href="/wiki/Music_hall">music hall</a> singer, ...
```

The user agent can choose any of the given resources depending on the user's screen's pixel density, zoom level, and possibly other factors such as the user's network conditions.

For backwards compatibility with older user agents that don't yet understand the [srcset](#) attribute, one of the URLs is specified in the [](#) element's [src](#) attribute. This will result in something useful (though perhaps lower-resolution than the user would like) being displayed even in older user agents. For new user agents, the [src](#) attribute participates in the resource selection, as if it was specified in [srcset](#) with a 1x descriptor.

The image's rendered size is given in the [width](#) and [height](#) attributes, which allows the user agent to allocate space for the image before it is downloaded.

Viewport-based selection

The [srcset](#) and [sizes](#) attributes can be used, using the w descriptor, to provide multiple images that only vary in their size (the smaller image is a scaled-down version of the bigger image).

EXAMPLE 347

In this example, a banner image takes up the entire [viewport](#) width (using appropriate CSS).

```
<h1></h1>
```

The user agent will calculate the effective pixel density of each image from the specified *w* descriptors and the specified rendered size in the *sizes* attribute. It can then choose any of the given resources depending on the user's screen's pixel density, zoom level, and possibly other factors such as the user's network conditions.

If the user's screen is 320 CSS pixels wide, this is equivalent to specifying `wolf-400.jpg 1.25x`, `wolf-800.jpg 2.5x`, `wolf-1600.jpg 5x`. On the other hand, if the user's screen is 1200 CSS pixels wide, this is equivalent to specifying `wolf-400.jpg 0.33x`, `wolf-800.jpg 0.67x`, `wolf-1600.jpg 1.33x`. By using the *w* descriptors and the *sizes* attribute, the user agent can choose the correct image source to download regardless of how large the user's device is.

For backwards compatibility, one of the URLs is specified in the `<img>` element's *src* attribute. In new user agents, the *src* attribute is ignored when the *srcset* attribute uses *w* descriptors.

In this example, the *sizes* attribute could be omitted because the default value is `100vw`.

EXAMPLE 348

In this example, the Web page has three layouts depending on the width of the [viewport](#). The narrow layout has one column of images (the width of each image is about 100%), the middle layout has two columns of images (the width of each image is about 50%), and the widest layout has three columns of images, and some page margin (the width of each image is about 33%). It breaks between these layouts when the [viewport](#) is 30em wide and 50em wide, respectively.

```

```

The *sizes* attribute sets up the layout breakpoints at 30em and 50em, and declares the image sizes between these breakpoints to be `100vw`, `50vw`, or `calc(33vw - 100px)`. These sizes do not necessarily have to match up exactly with the actual image width as specified in the CSS.

The user agent will pick a width from the *sizes* attribute, using the first item with a '`<media-condition>`' (the part in parentheses) that evaluates to true, or using the last item (`calc(33vw - 100px)`) if they all evaluate to false.

For example, if the [viewport](#) width is 29em, then `(max-width: 30em)` evaluates to true and `100vw` is used, so the image size, for the purpose of resource selection, is 29em. If the [viewport](#) width is instead 32em, then `(max-width: 30em)` evaluates to false, but `(max-width: 50em)` evaluates to true and `50vw` is used, so the image size, for the purpose of resource selection, is 16em (half the [viewport](#) width). Notice that the slightly wider [viewport](#) results in a smaller image because of the different layout.

The user agent can then calculate the effective pixel density and choose an appropriate resource similarly to the previous example.

Art direction-based selection

The `<picture>` element and the `<source>` element, together with the *media* attribute, can be used, to provide multiple images that vary the image content (for instance the smaller image might be a cropped version of the bigger image).

EXAMPLE 349

```
<picture>
  <source media="(min-width: 45em)" srcset="large.jpg">
  <source media="(min-width: 32em)" srcset="med.jpg">
  
</picture>
```

The user agent will choose the first [<source>](#) element for which the media query in the `media` attribute matches, and then choose an appropriate URL from its `srcset` attribute.

The rendered size of the image varies depending on which resource is chosen. To specify dimensions that the user agent can use before having downloaded the image, CSS can be used.

```
img { width: 300px; height: 300px }
@media (min-width: 32em) { img { width: 500px; height:300px } }
@media (min-width: 45em) { img { width: 700px; height:400px } }
```

EXAMPLE 350

This example combines [art direction](#)- and [device-pixel-ratio](#)-based selection. A banner that takes half the [viewport](#) is provided in two versions, one for wide screens and one for narrow screens.

```
<h1>
  <picture>
    <source media="(max-width: 500px)" srcset="banner-phone.jpeg, banner-phone-HD.jpeg 2x">
    
  </picture>
</h1>
```

[Image format-based selection](#)

The `type` attribute on the [<source>](#) element can be used, to provide multiple images in different formats.

EXAMPLE 351

```
<h2>From today's featured article</h2>
<picture>
  <source srcset="/uploads/100-marie-lloyd.webp" type="image/webp">
  <source srcset="/uploads/100-marie-lloyd.jpg" type="image/vnd.ms-photo">
  
</picture>
<p><b><a href="/wiki/Marie_Lloyd">Marie Lloyd</a></b> (1870–1922)
was an English <a href="/wiki/Music_hall">music hall</a> singer, ...</p>
```

In this example, the user agent will choose the first [<source>](#) that has a [type](#) attribute with a supported MIME type. If the user agent supports WebP images, the first [<source>](#) element will be chosen. If not, but the user agent does support JPEG XR images, the second [<source>](#) element will be chosen. If neither of those formats are supported, the [](#) element will be chosen.

§ 4.7.2. Dependencies

Media Queries [\[MEDIAQ\]](#)

[<media-condition>](#)

CSS Values and Units [\[CSS-VALUES\]](#)

[<length>](#)

CSS Syntax [\[CSS-SYNTAX-3\]](#)

[Parse a comma-separated list of component values](#)

[component value](#)

[<whitespace-token>](#)

§ 4.7.3. The `picture` element

Categories:

[Flow content.](#)

[Phrasing content.](#)

[Embedded content.](#)

Contexts in which this element can be used:

Where [embedded content](#) is expected.

Content model:

Zero or more [<source>](#) elements, followed by one [](#) element, optionally intermixed with [script-supporting elements](#).

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

None

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

DOM interface:

```
interface HTMLPictureElement : HTMLElement {};
```

The [<picture>](#) element is a container which provides multiples sources to its contained [](#) element to allow authors to declaratively control or give hints to the user agent about which image resource to use, based on the screen pixel density, [viewport](#) size, image format, and other factors. It [represents](#) its children.

NOTE:

The [<picture>](#) element is somewhat different from the similar-looking [video](#) and [<audio>](#) elements. While all of them contain [<source>](#) elements, the [<source>](#) element's `src` attribute has no meaning when the element is nested within a [<picture>](#) element, and the resource selection algorithm is different. As well, the [<picture>](#) element itself does not display anything; it merely provides a context for its contained [](#) element that enables it to choose from multiple [URLs](#).

§ 4.7.4. The `source` element when used with the `picture` element

Categories:

Same as for the [<source>](#) element.

Contexts in which this element can be used:

As a child of a [<picture>](#) element, before the [](#) element.

Content model:

Same as for the [<source>](#) element.

Tag omission in text/html:

No [end tag](#)

Content attributes:

[Global attributes](#)

`srcset` - Images to use in different situations (e.g., high-resolution displays, small monitors, etc)

sizes - Image sizes between breakpoints

media - Applicable media

type - Type of embedded resource

Allowed [ARIA role attribute](#) values:

None

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

[DOM interface](#):

```
partial interface HTMLSourceElement {
  attribute DOMString srcset;
  attribute DOMString sizes;
  attribute DOMString media;
};
```

The authoring requirements in this section only apply if the [<source>](#) element has a parent that is a [<picture>](#) element.

The [<source>](#) element allows authors to specify multiple alternative [source sets](#) for [](#) elements. It does not [represent](#) anything on its own.

The **srcset** content attribute must be present, and must consist of one or more [image candidate strings](#), each separated from the next by a U+002C COMMA character (.). If an [image candidate string](#) contains no descriptors and no [space characters](#) after the URL, the following [image candidate string](#), if there is one, must begin with one or more [space characters](#).

If the srcset attribute has any [image candidate strings](#) using a [width descriptor](#), the **sizes** content attribute must also be present, and the value must be a [valid source size list](#).

The **media** content attribute may also be present. If present, the value must contain a [valid media query list](#).

The **type** content attribute may also be present. If present, the value must be a [valid mime type](#). It gives the type of the images in the [source set](#), to allow the user agent to skip to the next [<source>](#) element if it does not support the given type.

NOTE:

If the type attribute is not specified, the user agent will not select a different [<source>](#) element if it finds that it does not support the image format after fetching it.

When a [<source>](#) element has a following sibling [<source>](#) element or [](#) element with a srcset attribute specified, it must have at least one of the following:

- A media attribute specified with a value that, after [stripping leading and trailing whitespace](#), is not the empty string and is not an [ASCII case-insensitive](#) match for the string "all".
- A type attribute specified.

The src attribute must not be present.

The IDL attributes [srcset](#), [sizes](#) and [media](#) must [reflect](#) the respective content attributes of the same name.

§ 4.7.5. The [img](#) element

[Categories](#):

[Flow content](#).

[Phrasing content](#).

[Embedded content](#).

[Form-associated element](#).

If the element has a usemap attribute: [interactive content](#).

[Palpable content](#).

[Contexts in which this element can be used](#):

Where [embedded content](#) is expected.

Content model:

Nothing

Tag omission in text/html:

No [end tag](#).

Content attributes:

[Global attributes](#)

alt - Replacement text for use when images are not available

src - Address of the resource

srcset - Images to use in different situations (e.g., high-resolution displays, small monitors, etc)

sizes - Image sizes between breakpoints

crossorigin - How the element handles crossorigin requests

usemap - Name of [image map](#) to use

ismap - Whether the image is a server-side image map

width - Horizontal dimension

height - Vertical dimension

Allowed [ARIA role attribute](#) values:

[‘presentation’](#) role only, for an [](#) element whose alt attribute’s value is empty (alt=""), otherwise [Any role value](#).

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
[NamedConstructor=Image(optional unsigned long width, optional unsigned long height)]
interface HTMLImageElement : HTMLElement {
    attribute DOMString alt;
    attribute DOMString src;
    attribute DOMString srcset;
    attribute DOMString sizes;
    attribute DOMString? crossorigin;
    attribute DOMString useMap;
    attribute boolean isMap;
    attribute unsigned long width;
    attribute unsigned long height;
    readonly attribute unsigned long naturalWidth;
    readonly attribute unsigned long naturalHeight;
    readonly attribute boolean complete;
    readonly attribute DOMString currentSrc;
};
```

An [](#) element represents an image and its [fallback content](#).

The image given by the **src** and **srcset** attributes, and any previous sibling [<source>](#) elements' **srcset** attributes if the parent is a [<picture>](#) element, is the embedded content; the value of the **alt** attribute is the [](#) element's [fallback content](#), and provides equivalent content for users and user agents who cannot process images or have image loading disabled.

Requirements for alternative representations of the image are described [in the next section](#).

The **src** attribute must be present, and must contain a [valid non-empty URL potentially surrounded by spaces](#) referencing a non-interactive, optionally animated, image resource that is neither paged nor scripted.

The **srcset** attribute may also be present. If present, its value must consist of one or more [image candidate strings](#), each separated from the next by a U+002C COMMA character (.). If an [image candidate string](#) contains no descriptors and no [space characters](#) after the URL, the following [image candidate string](#), if there is one, must begin with one or more [space characters](#).

An **image candidate string** consists of the following components, in order, with the further restrictions described below this list:

1. Zero or more [space characters](#).
2. A [valid non-empty URL](#) that does not start or end with a U+002C COMMA character (,), referencing a non-interactive, optionally animated, image resource that is neither paged nor scripted.
3. Zero or more [space characters](#).
4. Zero or one of the following:
 - A **width descriptor**, consisting of: a [space character](#), a [valid non-negative integer](#) giving a number greater than zero representing the **width descriptor value**, and a U+0077 LATIN SMALL LETTER W character.
 - A *pixel density descriptor*, consisting of: a [space character](#), a [valid floating-point number](#) giving a number greater than zero representing the *pixel density descriptor* value, and a U+0078 LATIN SMALL LETTER X character.
5. Zero or more [space characters](#).

There must not be an [image candidate string](#) for an element that has the same [width descriptor value](#) as another [image candidate string](#)'s [width descriptor value](#) for the same element.

There must not be an [image candidate string](#) for an element that has the same *pixel density descriptor* value as another [image candidate string](#)'s *pixel density descriptor* value for the same element. For the purpose of this requirement, an [image candidate string](#) with no descriptors is equivalent to an [image candidate string](#) with a 1x descriptor.

If a `<source>` element has a `sizes` attribute present or an `<img>` element has a `sizes` attribute present, all [image candidate strings](#) for that element must have the [width descriptor](#) specified.

If an [image candidate string](#) for a source or `<img>` element has the [width descriptor](#) specified, all other [image candidate strings](#) for that element must also have the [width descriptor](#) specified.

The specified width in an [image candidate string](#)'s [width descriptor](#) must match the [intrinsic width](#) in the resource given by the [image candidate string](#)'s URL, if it has an [intrinsic width](#).

NOTE:

The requirements above imply that images can be static bitmaps (e.g., PNGs, GIFs, JPEGs), single-page vector documents (single-page PDFs, XML files with an SVG root element), animated bitmaps (APNGs, animated GIFs), animated vector graphics (XML files with an SVG root element that use declarative SMIL animation), and so forth. However, these definitions preclude SVG files with script, multipage PDF files, interactive MNG files, HTML documents, plain text documents, and so forth. [\[PNG\]](#) [\[GIF\]](#) [\[JPEG\]](#) [\[PDF\]](#) [\[XML\]](#) [\[APNG\]](#) [\[SVG11\]](#) [\[MNG\]](#)

If the `srcset` attribute is present, the **sizes** attribute may also be present. If present, its value must be a [valid source size list](#).

A **valid source size list** is a string that matches the following grammar: [\[CSS-VALUES\]](#) [\[MEDIAQ\]](#)

```
<source-size-list> = <source-size># [ , <source-size-value> ]? | <source-size-value>
<source-size> = <media-condition> <source-size-value>
<source-size-value> = <length>
```

A `<source-size-value>` must not be negative.

NOTE:

Percentages are not allowed in a `<source-size-value>`, to avoid confusion about what it would be relative to. The `vw` unit can be used for sizes relative to the [viewport](#) width.

The `<img>` element must not be used as a layout tool. In particular, `img` elements should not be used to display transparent images, as such images rarely convey meaning and rarely add anything useful to the document.



The **crossorigin** attribute is a [CORS settings attribute](#). Its purpose is to allow images from third-party sites that allow cross-origin access to be used with [<canvas>](#).



An [](#) element has a **current request** and a **pending request**. The [current request](#) is initially set to a new [image request](#). The [pending request](#) is initially set to null. The [current request](#) is usually referred to as the [](#) element itself.

An **image request** has a **state**, **current URL** and **image data**.

An [image request](#)'s [state](#) is one of the following:

Unavailable

The user agent hasn't obtained any image data, or has obtained some or all of the image data but hasn't yet decoded enough of the image to get the image dimensions.

Partially available

The user agent has obtained some of the image data and at least the image dimensions are available.

Completely available

The user agent has obtained all of the image data and at least the image dimensions are available.

Broken

The user agent has obtained all of the image data that it can, but it cannot even decode the image enough to get the image dimensions (e.g., the image is corrupted, or the format is not supported, or no data could be obtained).

An [image request](#)'s [current URL](#) is initially the empty string.

An [image request](#)'s [image data](#) is the decoded image data.

When an [image request](#) is either in the [partially available](#) state or in the [completely available](#) state, it is said to be **available**.

An [image request](#) is initially [unavailable](#).

When an [](#) element is [available](#), it [provides a paint source](#) whose width is the image's [density-corrected intrinsic width](#) (if any), whose height is the image's [density-corrected intrinsic height](#) (if any), and whose appearance is the intrinsic appearance of the image.

In a [browsing context](#) where [scripting is disabled](#), user agents may obtain images immediately or on demand. In a [browsing context](#) where [scripting is enabled](#), user agents must obtain images immediately.

A user agent that obtains images immediately must immediately [update the image data](#) of an [](#) element, with the *restart animation* flag set if so stated, whenever that element is created or has experienced [relevant mutations](#).

A user agent that obtains images on demand must [update the image data](#) of an [](#) element whenever it needs the image data (i.e., on demand), but only if the [](#) element is in the [unavailable](#) state. When an [](#) element has experienced [relevant mutations](#), if the user agent only obtains images on demand, the [](#) element must return to the [unavailable](#) state.

The **relevant mutations** for an [](#) element are as follows:

- The element's `src`, `srcset`, `width`, or `sizes` attributes are set, changed, or removed.
- The element's `src` attribute is set to the same value as the previous value. This must set the *restart animation* flag for the [update the image data](#) algorithm.
- The element's `crossorigin` attribute's state is changed.
- The element is [inserted into](#) or [removed from](#) a picture parent element.
- The element's parent is a [<picture>](#) element and a source element is inserted as a previous sibling.
- The element's parent is a [<picture>](#) element and a [<source>](#) element that was a previous sibling is [removed](#).

- The element's parent is a [<picture>](#) element and a [<source>](#) element that is a previous sibling has its `srcset`, `sizes`, `media` or `type` attributes set, changed, or removed.
- The element's [adopting steps](#) are run.

Each [](#) element has a **last selected source**, which must initially be null.

Each [image request](#) has a **current pixel density**, which must initially be undefined.

When an [](#) element has a [current pixel density](#) that is not 1.0, the element's image data must be treated as if its resolution, in device pixels per CSS pixels, was the [current pixel density](#). The image's **density-corrected intrinsic width and height** are the [intrinsic width and height](#) after taking into account the [current pixel density](#).

EXAMPLE 352

For example, given a screen with 96 CSS pixels per CSS inch, if the [current pixel density](#) is 3.125, that means that there are $96 \times 3.125 = 300$ device pixels per CSS inch, and thus if the image data is 300x600, it has [intrinsic dimensions](#) of $300 \div 3.125 = 96$ CSS pixels by $600 \div 3.125 = 192$ CSS pixels. With a [current pixel density](#) of 2.0 (192 device pixels per CSS inch) and the same image data (300x600), the [intrinsic dimensions](#) would be 150x300.

Each [Document](#) object must have a **list of available images**. Each image in this list is identified by a tuple consisting of an [absolute URL](#), a [CORS settings attribute](#) mode, and, if the mode is not [No CORS](#), an [origin](#). Each image furthermore has an **ignore higher-layer caching** flag. User agents may copy entries from one [Document](#) object's [list of available images](#) to another at any time (e.g., when the [Document](#) is created, user agents can add to it all the images that are loaded in other [Documents](#)), but must not change the keys of entries copied in this way when doing so, and must unset the [ignore higher-layer caching](#) flag for the copied entry. User agents may also remove images from such lists at any time (e.g., to save memory). User agents must remove entries in the [list of available images](#) as appropriate given higher-layer caching semantics for the resource (e.g., the HTTP Cache-Control response header) when the [ignore higher-layer caching](#) flag is unset.

NOTE:

The [list of available images](#) is intended to enable synchronous switching when changing the `src` attribute to a URL that has previously been loaded, and to avoid re-downloading images in the same document even when they don't allow caching per HTTP. It is not used to avoid re-downloading the same image while the previous image is still loading.

EXAMPLE 353

For example, if a resource has the HTTP response header `Cache-Control: must-revalidate`, the user agent would remove it from the [list of available images](#) but could keep the image data separately, and use that if the server responds with a 204 No Content status.

When the user agent is to **update the image data** of an [](#) element, optionally with the *restart animations* flag set, it must run the following steps:

1. If the element's [node document](#) is not the [active document](#), then run these substeps:
 1. Continue running this algorithm [in parallel](#).
 2. Wait until the element's [node document](#) is the [active document](#).
 3. If another instance of this algorithm for this [](#) element was started after this instance (even if it aborted and is no longer running), then abort these steps.
 4. [Queue a microtask](#) to continue this algorithm.
2. If the user agent cannot support images, or its support for images has been disabled, then [abort the image request](#) for the [current request](#) and the [pending request](#), set [current request](#) to the [unavailable](#) state, let [pending request](#) be null, and abort these steps.
3. If the element does not use `srcset` or `picture` and it does not have a parent or it has a parent but it is not a [<picture>](#) element, and it has a `src` attribute specified and its value is not the empty string, let *selected source* be the value of the element's `src` attribute, and *selected pixel density* be 1.0. Otherwise, let *selected source* be null and *selected pixel density* be undefined.

4. Let the `<img>` element's [last selected source](#) be *selected source*.
5. If *selected source* is not null, run these substeps:
 1. [Parse](#) *selected source*, relative to the element, and let the result be *absolute URL*. If that is not successful, then abort these inner set of steps.
 2. Let *key* be a tuple consisting of the resulting [absolute URL](#), the `<img>` element's [crossorigin](#) attribute's mode, and, if that mode is not [No CORS](#), the [node document](#)'s [origin](#).
 3. If the [list of available images](#) contains an entry for *key*, run these subsubsteps:
 1. Set the [ignore higher-layer caching](#) flag for that entry.
 2. [Abort the image request](#) for the [current request](#) and the [pending request](#).
 3. Let [pending request](#) be null.
 4. Let [current request](#) be a new [image request](#) whose [image data](#) is that of the entry and whose state is set to the [completely available](#) state.
 5. Update the presentation of the image appropriately.
 6. Let the [current request](#)'s [current pixel density](#) be *selected pixel density*.
 7. [Queue a task](#) to [restart the animation](#) if *restart animation* is set, change [current request](#)'s [current URL](#) to *absolute URL*, and then [fire a simple event](#) named `load` at the `<img>` element.
 8. Abort the [update the image data](#) algorithm.
6. [in parallel await a stable state](#), allowing the [task](#) that invoked this algorithm to continue. The [synchronous section](#) consists of all the remaining steps of this algorithm until the algorithm says the [synchronous section](#) has ended. (Steps in [synchronous sections](#) are marked with )
7.  If another instance of this algorithm for this `<img>` element was started after this instance (even if it aborted and is no longer running), then abort these steps.

NOTE:

Only the last instance takes effect, to avoid multiple requests when, for example, the `src`, `srcset`, and `crossorigin` attributes are all set in succession.

8.  Let *selected source* and *selected pixel density* be the URL and pixel density that results from [selecting an image source](#), respectively.
 9.  If *selected source* is null, run these substeps:
 1.  Set the [current request](#) to the [broken](#) state, [abort the image request](#) for the [current request](#) and the [pending request](#), and let [pending request](#) be null.
 2.  [Queue a task](#) to change the [current request](#)'s [current URL](#) to the empty string, and then, if the element has a `src` attribute or it uses `srcset` or `picture<`, [fire a simple event](#) named `error` at the `<img>` element.
 3.  Abort this algorithm.
 10.  [Queue a task](#) to [fire a progress event](#) named `loadstart` at the `<img>` element.
-  [Parse](#) *selected source*, relative to the element's [node document](#), and let *absolute URL* be the [resulting URL string](#). If that is not successful, run these substeps:
1.  [Abort the image request](#) for the [current request](#) and the [pending request](#).
 2.  Set the [current request](#) to the [broken](#) state.
 3.  Let [pending request](#) be null.
 4.  [Queue a task](#) to change the [current request](#)'s [current URL](#) to *selected source*, [fire a simple event](#) named `error`

at the `<img>` element and then [fire a simple event](#) named `loadend` at the `<img>` element.

5. ⌚ Abort the update the image data algorithm.

11. ⌚ If the [pending request](#) is not null, and `absolute URL` is the same as the [pending request](#)'s [current URL](#), then abort these steps.

⌚ If `absolute URL` is the same as the [current request](#)'s [current URL](#), and [current request](#) is in the [partially available](#) state, then [abort the image request](#) for the [pending request](#), [queue a task](#) to [restart the animation](#) if [restart animation](#) is set, and abort these steps.

⌚ If the [pending request](#) is not null, [abort the image request](#) for the [pending request](#).

⌚ Let `image request` be a new [image request](#) whose [current URL](#) is `absolute URL`.

⌚ If [current request](#) is in the [unavailable](#) state or the [broken](#) state, let the [current request](#) be `image request`. Otherwise, let the [pending request](#) be `image request`.

⌚ Let `request` be the result of [creating a potential-CORS request](#) given `absolute URL` and the current state of the element's [crossorigin](#) content attribute.

⌚ Set `request`'s [client](#) to the element's [node document](#)'s Window object's [environment settings object](#) and [type](#) to `"image"`.

⌚ If the element uses `srcset` or `picture`, set `request`'s [initiator](#) to `"imageset"`.

⌚ Set `request`'s [same-origin data-URL flag](#).

⌚ [Fetch](#) `request`. Let this instance of the [fetching](#) algorithm be associated with `image request`.

The resource obtained in this fashion, if any, is `image request`'s [image data](#). It can be either [CORS-same-origin](#) or [CORS-cross-origin](#); this affects the [origin](#) of the image itself (e.g., when used on a `canvas`).

Fetching the image must [delay the load event](#) of the element's [node document](#) until the [task](#) that is [queued](#) by the [networking task source](#) once the resource has been fetched (defined below) has been run.

⚠Warning! This, unfortunately, can be used to perform a rudimentary port scan of the user's local network (especially in conjunction with scripting, though scripting isn't actually necessary to carry out such an attack). User agents may implement [cross-origin](#) access control policies that are stricter than those described above to mitigate this attack, but unfortunately such policies are typically not compatible with existing Web content.

If the resource is [CORS-same-origin](#), each [task](#) that is [queued](#) by the [networking task source](#) while the image is being fetched, if `image request` is the [current request](#), must [fire a progress event](#) named `progress` at the `<img>` element.

12. End the [synchronous section](#), continuing the remaining steps [in parallel](#), but without missing any data from fetching.

13. As soon as possible, jump to the first applicable entry from the following list:

↪ **If the resource type is `multipart/x-mixed-replace`**

The next [task](#) that is [queued](#) by the [networking task source](#) while the image is being fetched must run the following steps:

1. If `image request` is the [pending request](#) and at least one body part has been completely decoded, [abort the image request](#) for the [current request](#), [upgrade the pending request to the current request](#).
2. Otherwise, if `image request` is the [pending request](#) and the user agent is able to determine that `image request`'s image is corrupted in some fatal way such that the image dimensions cannot be obtained, [abort the image request](#) for the [current request](#), [upgrade the pending request to the current request](#) and set the [current request](#)'s state to [broken](#).
3. Otherwise, if `image request` is the [current request](#), it is in the [unavailable](#) state, and the user agent is able to determine `image request`'s image's width and height, set the [current request](#)'s state to [partially](#)

[available](#).

4. Otherwise, if *image request* is the [current request](#), it is in the [unavailable](#) state, and the user agent is able to determine that *image request*'s image is corrupted in some fatal way such that the image dimensions cannot be obtained, set the [current request](#)'s state to [broken](#).

Each [task](#) that is [queued](#) by the [networking task source](#) while the image is being fetched must update the presentation of the image, but as each new body part comes in, it must replace the previous image. Once one body part has been completely decoded, the user agent must set the `<img>` element to the [completely available](#) state and [queue a task](#) to [fire a simple event](#) named load at the `<img>` element.

NOTE:

The progress and loadend events are not fired for multipart/x-mixed-replace image streams.

↪ **If the resource type and data corresponds to a supported image format, as described below**

The next [task](#) that is [queued](#) by the [networking task source](#) while the image is being fetched must run the following steps:

1. If the user agent is able to determine *image request*'s image's width and height, and *image request* is [pending request](#), set *image request*'s state to [partially available](#).
2. Otherwise, if the user agent is able to determine *image request*'s image's width and height, and *image request* is [current request](#), update the `<img>` element's presentation appropriately and set *image request*'s state to [partially available](#).
3. Otherwise, if the user agent is able to determine that *image request*'s image is corrupted in some fatal way such that the image dimensions cannot be obtained, and *image request* is [pending request](#), [abort the image request](#) for the [current request](#) and the [pending request](#), [upgrade the pending request to the current request](#), set [current request](#) to the [broken](#) state, [fire a simple event](#) named error at the `<img>` element, [fire a simple event](#) named loadend at the `<img>` element, and abort these steps.
4. Otherwise, if the user agent is able to determine that *image request*'s image is corrupted in some fatal way such that the image dimensions cannot be obtained, and *image request* is [current request](#), [abort the image request](#) for *image request*, [fire a simple event](#) named error at the `<img>` element, [fire a simple event](#) named loadend at the `<img>` element, and abort these steps.

That [task](#), and each subsequent [task](#), that is [queued](#) by the [networking task source](#) while the image is being fetched, if *image request* is the [current request](#), must update the presentation of the image appropriately (e.g., if the image is a progressive JPEG, each packet can improve the resolution of the image).

Furthermore, the last [task](#) that is [queued](#) by the [networking task source](#) once the resource has been fetched must additionally run these steps:

1. If *image request* is the [pending request](#), [abort the image request](#) for the [current request](#), [upgrade the pending request to the current request](#) and update the `<img>` element's presentation appropriately.
2. Set *image request* to the [completely available](#) state.
3. Add the image to the [list of available images](#) using the key *key*, with the [ignore higher-layer caching](#) flag set.
4. [Fire a progress event or simple event](#) named load at the `<img>` element, depending on the resource in *image request*.
5. [Fire a progress event or simple event](#) named loadend at the `<img>` element, depending on the resource in *image request*.

↪ **Otherwise**

The image data is not in a supported file format; the user agent must set *image request* to the [broken](#) state, [abort the image request](#) for the [current request](#) and the [pending request](#), [upgrade the pending request to the current request](#) if *image request* is the [pending request](#), and then [queue a task](#) to first [fire a simple event](#) named error at the `<img>` element and then [fire a simple event](#) named loadend at the `<img>` element.

To **abort the image request** for an [image request](#) *image request* means to run the following steps:

1. Forget *image request*'s [image data](#), if any.
2. Abort any instance of the [fetching](#) algorithm for *image request*, discarding any pending tasks generated by that algorithm.

To **upgrade the pending request to the current request** for an [](#) element means to run the following steps:

1. Let the [](#) element's [current request](#) be the [pending request](#).
2. Let the [](#) element's [pending request](#) be null.

To **fire a progress event or simple event** named *type* at an element *e*, depending on resource *r*, means to [fire a progress event](#) named *type* at *e* if *r* is [CORS-same-origin](#), and otherwise [fire a simple event](#) named *type* at *e*.

While a user agent is running the above algorithm for an element *x*, there must be a strong reference from the element's [node document](#) to the element *x*, even if that element is not in its Document.

An [img](#) element is said to **use srcset or picture** if it has a [srcset](#) attribute specified or if it has a parent that is a picture element.

When an [](#) element is in the [completely available](#) state *and* the user agent can decode the media data without errors, then the [](#) element is said to be **fully decodable**.

Whether the image is fetched successfully or not (e.g., whether the response status was an [ok status](#)) must be ignored when determining the image's type and whether it is a valid image.

NOTE:

This allows servers to return images with error responses, and have them displayed.

The user agent should apply the [image sniffing rules](#) to determine the type of the image, with the image's [associated Content-Type headers](#) giving the *official type*. If these rules are not applied, then the type of the image must be the type given by the image's [associated Content-Type headers](#).

User agents must not support non-image resources with the [](#) element (e.g., XML files whose root element is an HTML element). User agents must not run executable code (e.g., scripts) embedded in the image resource. User agents must only display the first page of a multipage resource (e.g., a PDF file). User agents must not allow the resource to act in an interactive fashion, but should honor any animation in the resource.

This specification does not specify which image types are to be supported.



An [](#) element is associated with a [source set](#).

A **source set** is an ordered set of zero or more [image sources](#) and a [source size](#).

An **image source** is a [URL](#), and optionally either a density descriptor, or a [width descriptor](#).

A **source size** is a `<source-size-value>`. When a [source size](#) has a unit relative to the [viewport](#), it must be interpreted relative to the [](#) element's document's [viewport](#). Other units must be interpreted the same as in Media Queries. [\[MEDIAQ\]](#)

When asked to **select an image source** for a given [](#) element *el*, user agents must do the following:

1. [Update the source set](#) for *el*.
2. If *el*'s [source set](#) is empty, return null as the URL and undefined as the pixel density and abort these steps.
3. Otherwise, take *el*'s [source set](#) and let it be *source set*.
4. If an entry *b* in *source set* has the same associated density descriptor as an earlier entry *a* in *source set*, then remove entry *b*. Repeat this step until none of the entries in *source set* have the same associated density descriptor as an earlier entry.

5. In a user agent-specific manner, choose one [image source](#) from *source set*. Let this be *selected source*.
6. Return *selected source* and its associated pixel density.

When asked to **update the source set** for a given `<img>` element *el*, user agents must do the following:

1. Set *el*'s [source set](#) to an empty [source set](#).
2. If *el* has a parent node and that is a `<picture>` element, let *elements* be an array containing *el*'s parent node's child elements, retaining relative order. Otherwise, let *elements* be array containing only *el*.
3. If *el* has a *width* attribute, and parsing that attribute's value using the [rules for parsing dimension values](#) doesn't generate an error or a percentage value, then let *width* be the returned integer value. Otherwise, let *width* be null.
4. Iterate through *elements*, doing the following for each item *child*:
 1. If *child* is *el*:
 1. If *child* has a *srcset* attribute, [parse *child*'s srcset attribute](#) and let the returned [source set](#) be *source set*. Otherwise, let *source set* be an empty [source set](#).
 2. [Parse *child*'s sizes attribute](#) with the fallback width *width*, and let *source set*'s [source size](#) be the returned value.
 3. If *child* has a *src* attribute whose value is not the empty string and *source set* does not contain an [image source](#) with a density descriptor value of 1, and no [image source](#) with a [width descriptor](#), append *child*'s *src* attribute value to *source set*.
 4. [Normalize the source densities](#) of *source set*.
 5. Let *el*'s [source set](#) be *source set*.
 6. Abort this algorithm.
 2. If *child* is not a `<source>` element, continue to the next child. Otherwise, *child* is a `<source>` element.
 3. If *child* does not have a *srcset* attribute, continue to the next child.
 4. [Parse *child*'s srcset attribute](#) and let the returned [source set](#) be *source set*.
 5. If *source set* has zero [image sources](#), continue to the next child.
 6. If *child* has a *media* attribute, and its value does not [match the environment](#), continue to the next child.
 7. [Parse *child*'s sizes attribute](#) with the fallback width *width*, and let *source set*'s [source size](#) be the returned value.
 8. If *child* has a *type* attribute, and its value is an unknown or unsupported [MIME type](#), continue to the next child.
 9. [Normalize the source densities](#) of *source set*.
 10. Let *el*'s [source set](#) be *source set*.
 11. Abort this algorithm.

NOTE:

Each `<img>` element independently considers its previous sibling `<source>` elements plus the `<img>` element itself for selecting an [image source](#), ignoring any other (invalid) elements, including other `<img>` elements in the same `<picture>` element, or `<source>` elements that are following siblings of the relevant `<img>` element.

When asked to **parse a srcset attribute** from an element, parse the value of the element's *srcset* attribute as follows:

1. Let *input* be the value passed to this algorithm.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.

3. Let `candidates` be an initially empty [source set](#).
4. *Splitting loop:* [Collect a sequence of characters](#) that are [space characters](#) or U+002C COMMA characters. If any U+002C COMMA characters were collected, that is a [parse error](#).
5. If `position` is past the end of `input`, return `candidates` and abort these steps.
6. [Collect a sequence of characters](#) that are not [space characters](#), and let that be `url`.
7. Let `descriptors` be a new empty list.
8. If `url` ends with a U+002C COMMA character (,), follow these substeps:
 1. Remove all trailing U+002C COMMA characters from `url`. If this removed more than one character, that is a [parse error](#).

Otherwise, follow these substeps:

1. *Descriptor tokenizer:* [Skip whitespace](#)
2. Let `current descriptor` be the empty string.
3. Let `state` be *in descriptor*.
4. Let `c` be the character at `position`. Do the following depending on the value of `state`. For the purpose of this step, "EOF" is a special character representing that `position` is past the end of `input`.

↪ *In descriptor*

Do the following, depending on the value of `c`:

↪ [Space character](#)

If `current descriptor` is not empty, append `current descriptor` to `descriptors` and let `current descriptor` be the empty string. Set `state` to *after descriptor*.

↪ U+002C COMMA (,)

Advance `position` to the next character in `input`. If `current descriptor` is not empty, append `current descriptor` to `descriptors`. Jump to the step labeled *descriptor parser*.

↪ U+0028 LEFT PARENTHESIS (()

Append `c` to `current descriptor`. Set `state` to *in parens*.

↪ EOF

If `current descriptor` is not empty, append `current descriptor` to `descriptors`. Jump to the step labeled *descriptor parser*.

↪ Anything else

Append `c` to `current descriptor`.

↪ *In parens*

Do the following, depending on the value of `c`:

↪ U+0029 RIGHT PARENTHESIS ())

Append `c` to `current descriptor`. Set `state` to *in descriptor*.

↪ EOF

Append `current descriptor` to `descriptors`. Jump to the step labeled *descriptor parser*.

↪ Anything else

Append `c` to `current descriptor`.

↪ *After descriptor*

Do the following, depending on the value of `c`:

↪ [Space character](#)

Stay in this state.

↪ EOF

Jump to the step labeled *descriptor parser*.

↪ **Anything else**

Set *state* to *in descriptor*. Set *position* to the *previous* character in *input*.

Advance *position* to the next character in *input*. Repeat this substep.

NOTE:

In order to be compatible with future additions, this algorithm supports multiple descriptors and descriptors with parens.

9. *Descriptor parser*: Let *error* be *no*.

10. Let *width* be *absent*.

11. Let *density* be *absent*.

12. Let *future-compat-h* be *absent*.

13. For each descriptor in *descriptors*, run the appropriate set of steps from the following list:

↪ **If the descriptor consists of a [valid non-negative integer](#) followed by a U+0077 LATIN SMALL LETTER W character**

1. If the user agent does not support the *sizes* attribute, let *error* be *yes*.

NOTE:

A conforming user agent will support the *sizes* attribute. However, user agents typically implement and ship features in an incremental manner in practice.

2. If *width* and *density* are not both *absent*, then let *error* be *yes*.

3. Apply the [rules for parsing non-negative integers](#) to the descriptor. If the result is zero, let *error* be *yes*. Otherwise, let *width* be the result.

↪ **If the descriptor consists of a [valid floating-point number](#) followed by a U+0078 LATIN SMALL LETTER X character**

1. If *width*, *density* and *future-compat-h* are not all *absent*, then let *error* be *yes*.

2. Apply the [rules for parsing floating-point number values](#) to the descriptor. If the result is less than zero, let *error* be *yes*. Otherwise, let *density* be the result.

NOTE:

If *density* is zero, the [intrinsic dimensions](#) will be infinite. User agents are expected to have limits in how big images can be rendered, which is allowed by the [hardware limitations](#) clause.

↪ **If the descriptor consists of a [valid non-negative integer](#) followed by a U+0068 LATIN SMALL LETTER H character**

This is a [parse error](#).

1. If *future-compat-h* and *density* are not both *absent*, then let *error* be *yes*.

2. Apply the [rules for parsing non-negative integers](#) to the descriptor. If the result is zero, let *error* be *yes*. Otherwise, let *future-compat-h* be the result.

↪ **Anything else**

Let *error* be *yes*.

14. If *future-compat-h* is not *absent* and *width* is *absent*, let *error* be *yes*.

15. If *error* is still *no*, then append a new [image source](#) to *candidates* whose URL is *url*, associated with a width *width* if not *absent* and a pixel density *density* if not *absent*. Otherwise, there is a [parse error](#).

16. Return to the step labeled *splitting loop*.

When asked to **parse a sizes attribute** from an element, [parse a comma-separated list of component values](#) from the value of the element's sizes attribute (or the empty string, if the attribute is absent), and let *unparsed sizes list* be the result.

[\[CSS-SYNTAX-3\]](#)

For each *unparsed size* in *unparsed sizes list*:

1. Remove all consecutive [‘<whitespace-token>’](#)s from the end of *unparsed size*. If *unparsed size* is now empty, that is a [parse error](#); continue to the next iteration of this algorithm.
2. If the last [component value](#) in *unparsed size* is a valid non-negative [<source-size-value>](#), let *size* be its value and remove the [component value](#) from *unparsed size*. Any CSS function other than the `calc()` function is invalid. Otherwise, there is a [parse error](#); continue to the next iteration of this algorithm.
3. Remove all consecutive [‘<whitespace-token>’](#)s from the end of *unparsed size*. If *unparsed size* is now empty, return *size* and exit this algorithm. If this was not the last item in *unparsed sizes list*, that is a [parse error](#).
4. Parse the remaining [component values](#) in *unparsed size* as a [‘<media-condition>’](#). If it does not parse correctly, or it does parse correctly but the [‘<media-condition>’](#) evaluates to false, continue to the next iteration of this algorithm.
[\[MEDIAQ\]](#)
5. Return *size* and exit this algorithm.

If the above algorithm exhausts *unparsed sizes list* without returning a *size* value, follow these steps:

1. If *width* is not null, return a [<length>](#) with the value *width* and the unit px.
2. Return 100vw.

A [parse error](#) for the algorithms above indicates a non-fatal mismatch between input and requirements. User agents are encouraged to expose [parse errors](#) somehow.

NOTE:

While a [valid source size list](#) only contains a bare [<source-size-value>](#) (without an accompanying [<media-condition>](#)) as the last entry in the [<source-size-list>](#), the parsing algorithm technically allows such at any point in the list, and will accept it immediately as the size if the preceding entries in the list weren't used. This is to enable future extensions, and protect against simple author errors such as a final trailing comma.

An [image source](#) can have a density descriptor, a [width descriptor](#), or no descriptor at all accompanying its URL. Normalizing a [source set](#) gives every [image source](#) a density descriptor.

When asked to **normalize the source densities** of a [source set](#) *source set*, the user agent must do the following:

1. Let *source size* be *source set*'s [source size](#).
2. For each [image source](#) in *source set*:
 1. If the [image source](#) has a density descriptor, continue to the next [image source](#).
 2. Otherwise, if the [image source](#) has a [width descriptor](#), replace the [width descriptor](#) with a density descriptor with a value of the [width descriptor](#) divided by the [source size](#) and a unit of x.

NOTE:

If the [source size](#) is zero, the density would be infinity, which results in the [intrinsic dimensions](#) being zero by zero.

3. Otherwise, give the [image source](#) a density descriptor of 1x.

The user agent may at any time run the following algorithm to update an `img` element's image in order to react to changes in the environment. (User agents are *not required* to ever run this algorithm; for example, if the user is not looking at the page any more, the user agent might want to wait until the user has returned to the page before determining which image to use, in case the environment changes again in the meantime.)

NOTE:

User agents are encouraged to run this algorithm in particular when the user changes the [viewport](#)'s size (e.g., by resizing the window or changing the page zoom), and when an [](#) element is [inserted into a document](#), so that the [density-corrected intrinsic width and height](#) match the new [viewport](#), and so that the correct image is chosen when [art direction](#) is involved.

1. [in parallel](#) [await a stable state](#). The [synchronous section](#) consists of all the remaining steps of this algorithm until the algorithm says the [synchronous section](#) has ended. (Steps in [synchronous sections](#) are marked with )
2.  If the [](#) element does not [use srcset or picture](#), its [node document](#) is not the [active document](#), has image data whose resource type is [multipart/x-mixed-replace](#), or the [pending request](#) is not null, then abort this algorithm.
3.  Let *selected source* and *selected pixel density* be the URL and pixel density that results from [selecting an image source](#), respectively.
4.  If *selected source* is null, then abort these steps.
5.  If *selected source* and *selected pixel density* are the same as the element's [last selected source](#) and [current pixel density](#), then abort these steps.
6.  [Parse](#) *selected source*, relative to the element's [node document](#), and let *absolute URL* be the [resulting URL string](#). If that is not successful, abort these steps.
7.  Let *corsAttributeState* be the state of the element's [crossorigin](#) content attribute.
8.  Let *origin* be the [origin](#) of the [](#) element's [node document](#).
9.  Let *client* be the [](#) element's [node document](#)'s Window object's [environment settings object](#).
10.  Let *key* be a tuple consisting of *absolute URL*, *corsAttributeState*, and, if *corsAttributeState* is not [No CORS](#), *origin*.
11.  Let *image request* be a new [image request](#) whose [current URL](#) is *absolute URL*.
12.  Let the element's [pending request](#) be *image request*.
13. End the [synchronous section](#), continuing the remaining steps [in parallel](#).
14. If the [list of available images](#) contains an entry for *key*, then set *image request*'s [image data](#) to that of the entry. Continue to the next step.

Otherwise, run these substeps:
 1. Let *request* be the result of [creating a potential-CORS request](#) given *absolute URL* and *corsAttributeState*.
 2. Set *request*'s [client](#) to *client*, [type](#) to "image", and set *request*'s [synchronous flag](#).
 3. Let *response* be the result of [fetching request](#).
 4. If *response*'s unsafe response is a network error or if the image format is unsupported (as determined by applying the [image sniffing rules](#), again as mentioned earlier), or if the user agent is able to determine that *image request*'s image is corrupted in some fatal way such that the image dimensions cannot be obtained, or if the resource type is [multipart/x-mixed-replace](#), then let [pending request](#) be null and abort these steps.
 5. Otherwise, *response*'s unsafe response is *image request*'s [image data](#). It can be either [CORS-same-origin](#) or [CORS-cross-origin](#); this affects the [origin](#) of the image itself (e.g., when used on a canvas).
15. [Queue a task](#) to run the following substeps:
 1. If the [](#) element has experienced [relevant mutations](#) since this algorithm started, then let [pending request](#) be null and abort these steps.
 2. Let the [](#) element's [last selected source](#) be *selected source* and the [](#) element's [current pixel density](#) be *selected pixel density*.

3. Set *image request* to the [completely available](#) state.
4. Add the image to the [list of available images](#) using the key *key*, with the [ignore higher-layer caching](#) flag set.
5. [Upgrade the pending request to the current request](#).
6. Update the `<img>` element's presentation appropriately.
7. [Fire a simple event](#) named *load* at the `<img>` element.



The [task source](#) for the [tasks queued](#) by algorithms in this section is the [DOM manipulation task source](#).



What an `<img>` element represents depends on the *src* attribute and the *alt* attribute.

↪ **If the *src* attribute is set and the *alt* attribute is set to the empty string**

The image is either decorative or supplemental to the rest of the content, redundant with some other information in the document.

If the image is [available](#) and the user agent is configured to display that image, then the element [represents](#) the element's image data.

Otherwise, the element [represents](#) nothing, and may be omitted completely from the rendering. User agents may provide the user with a notification that an image is present but has been omitted from the rendering.

↪ **If the *src* attribute is set and the *alt* attribute is set to a value that isn't empty**

The image is a key part of the content; the *alt* attribute gives a textual equivalent or replacement for the image.

If the image is [available](#) and the user agent is configured to display that image, then the element [represents](#) the element's image data.

Otherwise, the element [represents](#) the text given by the *alt* attribute. User agents may provide the user with a notification that an image is present but has been omitted from the rendering.

↪ **If the *src* attribute is set and the *alt* attribute is not**

There is no textual equivalent of the image available.

If the image is [available](#) and the user agent is configured to display that image, then the element [represents](#) the element's image data.

Otherwise, the user agent should display some sort of indicator that there is an image that is not being rendered, and may, if requested by the user, or if so configured, or when required to provide contextual information in response to navigation, provide caption information for the image, derived as follows:

1. If the image is a descendant of a `<figure>` element that has a child `<figcaption>` element, and, ignoring the `<figcaption>` element and its descendants, the `<figure>` element has no Text node descendants other than [inter-element whitespace](#), and no [embedded content](#) descendant other than the `<img>` element, then the contents of the first such `<figcaption>` element are the caption information; abort these steps.
2. There is no caption information.

↪ **If the *src* attribute is not set and either the *alt* attribute is set to the empty string or the *alt* attribute is not set at all**

The element [represents](#) nothing.

↪ **Otherwise**

The element [represents](#) the text given by the *alt* attribute.

The *alt* attribute does not represent advisory information. User agents must not present the contents of the *alt* attribute in

the same way as content of the `title` attribute.

User agents may always provide the user with the option to display any image, or to prevent any image from being displayed. User agents may also apply heuristics to help the user make use of the image when the user is unable to see it, e.g., due to a visual disability or because they are using a text terminal with no graphics capabilities. Such heuristics could include, for instance, optical character recognition (OCR) of text found within the image.

⚠Warning! While user agents are encouraged to repair cases of missing `alt` attributes, authors must not rely on such behavior. [Requirements for providing text to act as an alternative for images](#) are described in detail below.

The contents of `<img>` elements, if any, are ignored for the purposes of rendering.



The `usemap` attribute, if present, can indicate that the image has an associated [image map](#).

The `ismap` attribute, when used on an element that is a descendant of an `<a>` element with an `href` attribute, indicates by its presence that the element provides access to a server-side image map. This affects how events are handled on the corresponding `<a>` element.

The `ismap` attribute is a [boolean attribute](#). The attribute must not be specified on an element that does not have an ancestor `<a>` element with an `href` attribute.

NOTE:

The `usemap` and `ismap` attributes can result in confusing behavior when used together with `<source>` elements with the `media` attribute specified in a `<picture>` element.

The `<img>` element supports [dimension attributes](#).

The `alt`, `src`, `srcset` and `sizes` IDL attributes must [reflect](#) the respective content attributes of the same name.

The `crossOrigin` IDL attribute must [reflect](#) the `crossorigin` content attribute.

The `useMap` IDL attribute must [reflect](#) the `usemap` content attribute.

The `isMap` IDL attribute must [reflect](#) the `ismap` content attribute.

This definition is non-normative. Implementation requirements are given below this definition.

`image . width` [= *value*]

`image . height` [= *value*]

These attributes return the actual rendered dimensions of the image, or zero if the dimensions are not known.

They can be set, to change the corresponding content attributes.

`image . naturalWidth`

`image . naturalHeight`

These attributes return the intrinsic dimensions of the image, or zero if the dimensions are not known.

`image . complete`

Returns true if the image has been completely downloaded or if no image is specified; otherwise, returns false.

`image . currentSrc`

Returns the image's [absolute URL](#).

`image = new Image( [ width [, height ] ] )`

Returns a new `<img>` element, with the `width` and `height` attributes set to the values passed in the relevant arguments, if applicable.

The IDL attributes **width** and **height** must return the rendered width and height of the image, in CSS pixels, if the image is [being rendered](#), and is being rendered to a visual medium; or else the [density-corrected intrinsic width and height](#) of the image, in CSS pixels, if the image has [intrinsic dimensions](#) and is *available* but not being rendered to a visual medium; or else 0, if the image is not *available* or does not have [intrinsic dimensions](#). [\[CSS-2015\]](#)

On setting, they must act as if they [reflected](#) the respective content attributes of the same name.

The IDL attributes **naturalWidth** and **naturalHeight** must return the [density-corrected intrinsic width and height](#) of the image, in CSS pixels, if the image has [intrinsic dimensions](#) and is *available*, or else 0. [\[CSS-2015\]](#)

The IDL attribute **complete** must return true if any of the following conditions is true:

- Both the `src` attribute and the `srcset` attribute are omitted.
- The `srcset` attribute is omitted and the `src` attribute's value is the empty string.
- The final [task](#) that is [queued](#) by the [networking task source](#) once the resource has been fetched has been [queued](#).
- The `<img>` element is [completely available](#).
- The `<img>` element is [broken](#).

Otherwise, the attribute must return false.

NOTE:

The value of `complete` can thus change while a [script](#) is executing.

The **currentSrc** IDL attribute must return the `<img>` element's [current request](#)'s [current URL](#).

A constructor is provided for creating `HTMLImageElement` objects (in addition to the factory methods from DOM such as `createElement()`): `Image(width, height)`. When invoked as a constructor, this must return a new `HTMLImageElement` object (a new `<img>` element). If the *width* argument is present, the new object's `width` content attribute must be set to *width*. If the *height* argument is also present, the new object's `height` content attribute must be set to *height*. The element's [node document](#) must be the [active document](#) of the [browsing context](#) of the `Window` object on which the interface object of the invoked constructor is found.

§ 4.7.5.1. Requirements for providing text to act as an alternative for images

[Text alternatives](#), [\[WCAG20\]](#) are a primary way of making visual information accessible, because they can be rendered through many sensory modalities (for example, visual, auditory or tactile) to match the needs of the user. Providing text alternatives allows the information to be rendered in a variety of ways by a variety of user agents. For example, a person who cannot see a picture can hear the text alternative read aloud using synthesized speech.

The `alt` attribute on images is a very important accessibility attribute. Authoring useful `alt` attribute content requires the author to carefully consider the context in which the image appears and the function that image may have in that context.

The guidance included here addresses the most common ways authors use images. Additional guidance and techniques are available in [Resources on Alternative Text for Images](#).

§ 4.7.5.1.1. EXAMPLES OF SCENARIOS WHERE USERS BENEFIT FROM TEXT ALTERNATIVES FOR IMAGES

- They have a very slow connection and are browsing with images disabled.
- They have a vision impairment and use text to speech software.
- They have a cognitive impairment and use text to speech software.
- They are using a text-only browser.
- They are listening to the page being read out by a voice Web browser.
- They have images disabled to save on download costs.
- They have problems loading images or the source of an image is wrong.

§ 4.7.5.1.2. GENERAL GUIDELINES

Except where otherwise specified, the `alt` attribute must be specified and its value must not be empty; the value must be an appropriate functional replacement for the image. The specific requirements for the `alt` attribute content depend on the image's function in the page, as described in the following sections.

To determine an appropriate text alternative it is important to think about why an image is being included in a page. What is its purpose? Thinking like this will help you to understand what is important about the image for the intended audience. Every image has a reason for being on a page, because it provides useful information, performs a function, labels an interactive element, enhances aesthetics or is purely decorative. Therefore, knowing what the image is for, makes writing an appropriate text alternative easier.

§ 4.7.5.1.3. A LINK OR BUTTON CONTAINING NOTHING BUT AN IMAGE

When an `<a>` element that is a [hyperlink](#), or a `<button>` element, has no text content but contains one or more images, include text in the `alt` attribute(s) that together convey the purpose of the link or button.

EXAMPLE 354

In this example, a portion of an editor interface is displayed. Each button has an icon representing an action a user can take on content they are editing. For users who cannot view the images, the action names are included within the `alt` attributes of the images:



```
<ul>
  <li><button></button></li>
  <li><button></button></li>
  <li><button></button></li>
  <li><button></button></li>
  <li><button></button></li>
</ul>
```

EXAMPLE 355

In this example, a link contains a logo. The link points to the W3C web site **from an external site**. The text alternative is a brief description of the link target.



```
<a href="https://w3.org">
  
</a>
```

EXAMPLE 356

This example is the same as the previous example, except that the **link is on the W3C web site**. The text alternative is a brief description of the link target.



```
<a href="https://w3.org">

</a>
```

NOTE:

Depending on the context in which an image of a logo is used it could be appropriate to provide an indication, as part of the text alternative, that the image is a logo. Refer to section [§4.7.5.1.19 Logos, insignia, flags, or emblems](#).

EXAMPLE 357

In this example, a link contains a print preview icon. The link points to a version of the page with a print stylesheet applied. The text alternative is a brief description of the link target.



```
<a href="preview.html">

</a>
```

EXAMPLE 358

In this example, a button contains a search icon. The button submits a search form. The text alternative is a brief description of what the button does.



```
<button>

</button>
```

EXAMPLE 359

In this example, a company logo for the *PIP Corporation* has been split into the following two images, the first containing the word *PIP* and the second with the abbreviated word *CO*. The images are the sole content of a link to the *PIPCO* home page. In this case a brief description of the link target is provided. As the images are presented to the user as a single entity the text alternative *PIP CO home* is in the `alt` attribute of the first image.



```
<a href="pipco-home.html">

</a>
```

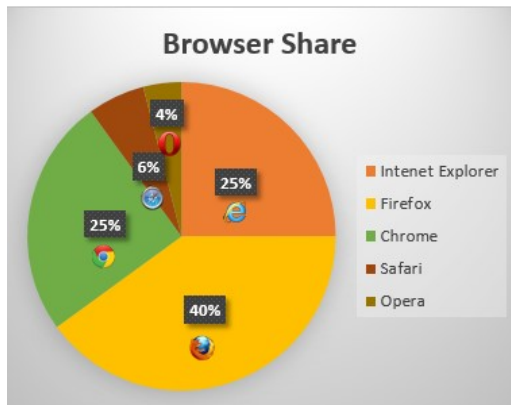
§ 4.7.5.1.4. GRAPHICAL REPRESENTATIONS: CHARTS, DIAGRAMS, GRAPHS, MAPS, ILLUSTRATIONS

Users can benefit when content is presented in graphical form, for example as a flowchart, a diagram, a graph, or a map

showing directions. Users who are unable to view the image also benefit when content presented in a graphical form is provided in a text-based format. Software agents that process text content, but cannot automatically process images (e.g. translation services, many search engines), also benefit from a text-based description.

EXAMPLE 360

In the following example we have an image of a pie chart, with text in the `alt` attribute representing the data shown in the pie chart:



```

```

EXAMPLE 361

In the case where an image repeats the previous paragraph in graphical form. The `alt` attribute content labels the image.

```
<p id="graph7">According to a recent study Firefox has a 40% browser share,
Internet Explorer has 25%, Chrome has 25%, Safari has 6% and Opera has 4%.</p>
<p></p>
```

It can be seen that when the image is not available, for example because the `src` attribute value is incorrect, the text alternative provides the user with a brief description of the image content:

According to a recent study Firefox has a 40% browser share, Internet Explorer has 25%, Chrome has 40%, Safari has 6% and Opera has 4%.

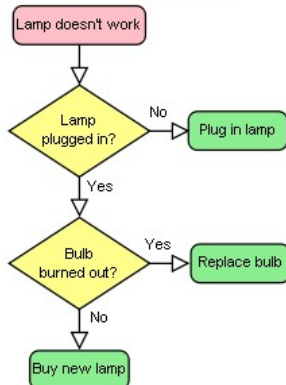
☒ Pie chart representing the data in the previous paragraph.

NOTE:

In cases where the text alternative is lengthy, more than a sentence or two, or would benefit from the use of structured markup, provide a brief description or label using the `alt` attribute, and an associated text alternative.

EXAMPLE 362

Here's an example of a flowchart image, with a short text alternative included in the `alt` attribute, in this case the text alternative is a description of the link target as the image is the sole content of a link. The link points to a description, within the same document, of the process represented in the flowchart.

Dealing with a broken lamp

```
<a href="#desc"></a>
```

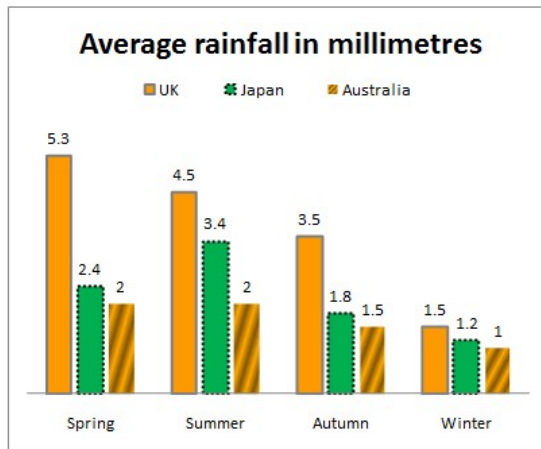
...

...

```
<div id="desc">
<h2>Dealing with a broken lamp</h2>
<ol>
<li>Check if it's plugged in, if not, plug it in.</li>
<li>If it still doesn't work; check if the bulb is burned out. If it is, replace the
bulb.</li>
<li>If it still doesn't work; buy a new lamp.</li>
</ol>
</div>
```

EXAMPLE 363

In this example, there is an image of a chart. It would be inappropriate to provide the information depicted in the chart as a plain text alternative in an `alt` attribute as the information is a data set. Instead a structured text alternative is provided below the image in the form of a data table using the data that is represented in the chart image.

**NOTE:**

Indications of the highest and lowest rainfall for each season have been included in the table, so trends easily identified in the chart are also available in the data table.

Average rainfall in millimetres by country and season.

	United Kingdom	Japan	Australia
Spring	5.3 (highest)	2.4	2 (lowest)
Summer	4.5 (highest)	3.4	2 (lowest)
Autumn	3.5 (highest)	1.8	1.5 (lowest)
Winter	1.5 (highest)	1.2	1 (lowest)

```

<figure>
<figcaption>Rainfall Data</figcaption>

<table id="table-4">
<caption>Rainfall in millimetres by Country and Season.</caption>
<tr><td><th scope="col">UK <th scope="col">Japan<th scope="col">Australia</tr>
<tr><th scope="row">Spring <td>5.5 (highest)<td>2.4 <td>2 (lowest)</tr>
<tr><th scope="row">Summer <td>4.5 (highest)<td>3.4<td>2 (lowest)</tr>
<tr><th scope="row">Autumn <td>3.5 (highest) <td>1.8 <td>1.5 (lowest)</tr>
<tr><th scope="row">Winter <td>1.5 (highest) <td>1.2 <td>1 lowest</tr>
</table>
</figure>

```

NOTE:

The `<figure>` element is used to group the Bar Chart image and data table. The `<figcaption>` element provides a caption for the grouped content.

NOTE:

For any of the examples in this section the details and summary elements could be used so that the text descriptions for the images are only displayed on demand:

EXAMPLE 364

► Dealing with a broken lamp

▼ Dealing with a broken lamp

1. Check if it's plugged in, if not, plug it in.
2. If it still doesn't work; check if the bulb is burned out. If it is, replace the bulb.
3. If it still doesn't work; buy a new lamp.

```
<figure>

<details>
<summary>Dealing with a broken lamp</summary>
<ol>
<li>Check if it's plugged in, if not, plug it in.</li>
<li>If it still doesn't work; check if the bulb is burned out. If it is, replace the
bulb.</li>
<li>If it still doesn't work; buy a new lamp.</li>
</ol>
</details>
</figure>
```

NOTE:

The `<details>` and `<summary>` elements are not currently well supported by browsers, until such times they are supported, if used, you will need to use scripting to provide the functionality. There are a number of scripted Polyfills and scripted custom controls available, in popular JavaScript UI widget libraries, which provide similar functionality.

§ 4.7.5.1.5. IMAGES OF TEXT

Sometimes, an image only contains text, and the purpose of the image is to display text using visual effects and /or fonts. It is *strongly* recommended that text styled using CSS be used, but if this is not possible, provide the same text in the `alt` attribute as is in the image.

EXAMPLE 365

This example shows an image of the text "Get Happy!" written in a fancy multi colored freehand style. The image makes up the content of a heading. In this case the text alternative for the image is "Get Happy!".



```
<h1></h1>
```

EXAMPLE 366

In this example we have an advertising image consisting of text, the phrase "The BIG sale" is repeated 3 times, each time the text gets smaller and fainter, the last line reads "...ends Friday" In the context of use, as an advertisement, it is recommended that the image's text alternative only include the text "The BIG sale" once as the repetition is for visual effect and the repetition of the text for users who cannot view the image is unnecessary and could be confusing.



```
<p></p>
```

NOTE:

In situations where there is also a photo or other graphic along with the image of text, ensure that the words in the image text are included in the text alternative, along with any other description of the image that conveys meaning to users who can view the image, so the information is also available to users who cannot view the image.

When an image is used to represent a character that cannot otherwise be represented in Unicode, for example gaiji, itaiji, or new characters such as novel currency symbols, the alternative text should be a more conventional way of writing the same thing, e.g., using the phonetic hiragana or katakana to give the character's pronunciation.

EXAMPLE 367

In this example from 1997, a new-fangled currency symbol that looks like a curly E with two bars in the middle instead of one is represented using an image. The alternative text gives the character's pronunciation.

Only € 5.99!

```
<p>Only 5.99!
```

An image should not be used if Unicode characters would serve an identical purpose. Only when the text cannot be directly represented using Unicode, e.g., because of decorations or because the character is not in the Unicode character set (as in the case of gaiji), would an image be appropriate.

NOTE:

If an author is tempted to use an image because their default system font does not support a given character, then Web Fonts are a better solution than images.

EXAMPLE 368

An illuminated manuscript might use graphics for some of its letters. The text alternative in such a situation is just the character that the image represents. nce upon a time and a long long time ago...

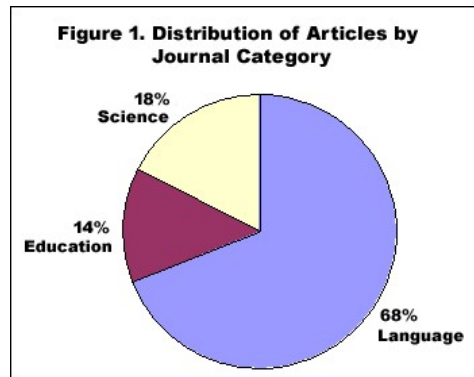
```
<p>nice upon a time and a long long time ago...
```

§ 4.7.5.1.6. IMAGES THAT INCLUDE TEXT

Sometimes, an image consists of a graphics such as a chart and associated text. In this case it is recommended that the text in the image is included in the text alternative.

EXAMPLE 369

Consider an image containing a pie chart and associated text. It is recommended wherever possible to provide any associated text as text, not an image of text. If this is not possible include the text in the text alternative along with the pertinent information conveyed in the image.



```
<p></p>
```

EXAMPLE 370

Here's another example of the same pie chart image, showing a short text alternative included in the `alt` attribute and a longer text alternative in text. The `figure` and `figcaption` elements are used to associate the longer text alternative with the image. The `alt` attribute is used to label the image.

```
<figure>

<figcaption><strong>Figure 1.</strong> Distribution of Articles by Journal Category.
Pie chart: Language=68%, Education=14% and Science=18%.</figcaption>
</figure>
```

NOTE:

The advantage of this method over the previous example is that the text alternative is available to all users at all times. It also allows structured mark up to be used in the text alternative, where as a text alternative provided using the `alt` attribute does not.

§ 4.7.5.1.7. IMAGES THAT ENHANCE THE THEMES OR SUBJECT MATTER OF THE PAGE CONTENT

An image that isn't discussed directly by the surrounding text but still has some relevance can be included in a page using the `<img>` element. Such images are more than mere decoration, they may augment the themes or subject matter of the page content and so still form part of the content. In these cases, it is recommended that a text alternative be provided.

EXAMPLE 371

Here is an example of an image closely related to the subject matter of the page content but not directly discussed. An image of a painting inspired by a poem, on a page reciting that poem. The following snippet shows an example. The image is a painting titled the "Lady of Shalott", it is inspired by the poem and its subject matter is derived from the poem. Therefore it is strongly recommended that a text alternative is provided. There is a short description of the content of the image in the alt attribute and a link below the image to a longer description located at the bottom of the document. At the end of the longer description there is also a link to further information about the painting.



```
<header>
<h1>The Lady of Shalott</h1>
<p>A poem by Alfred Lord Tennyson</p>
</header>


<p><a href="#des">Description of the painting</a>.</p>

<!-- Full Recitation of Alfred, Lord Tennyson's Poem. -->

...
...
...
<p id="des">The woman in the painting is wearing a flowing white dress. A large piece of
intricately
patterned fabric is draped over the side. In her right hand she holds the chain mooring the
boat. Her expression
is mournful. She stares at a crucifix lying in front of her. Beside it are three candles. Two
have blown out.
<a href="https://bit.ly/5HJvVZ">Further information about the painting</a>.</p>
```

EXAMPLE 372

This example illustrates the provision of a text alternative identifying an image as a photo of the main subject of a page.



Robin Berjon

What more needs to be said?

```

<h1>Robin Berjon</h1>
<p>What more needs to be said?</p>
```

In many cases, the image is actually just supplementary, and its presence merely reinforces the surrounding text. In these cases, the `alt` attribute must be present but its value must be the empty string.

In general, an image falls into this category if removing the image doesn't make the page any less useful, but including the image makes it a lot easier for users of visual browsers to understand the concept.

EXAMPLE 373

It is not always easy to write a useful text alternative for an image, another option is to provide a link to a description or further information about the image when one is available. In this example of the same image, there is a short text alternative included in the `alt` attribute, and there is a link after the image. The link points to a page containing [information about the painting](#).

The Lady of Shalott

A poem by Alfred Lord Tennyson.



[About this painting](#)

Full recitation of Alfred, Lord Tennyson's poem.

```
<header><h1>The Lady of Shalott</h1>
<p>A poem by Alfred Lord Tennyson</p></header>
<figure>

<p><a href="https://bit.ly/5HJVZ">About this painting.</a></p>
</figure>
<!-- Full Recitation of Alfred, Lord Tennyson's Poem. -->
```

§ 4.7.5.1.9. A PURELY DECORATIVE IMAGE THAT DOESN'T ADD ANY INFORMATION

Purely decorative images are visual enhancements, decorations or embellishments that provide no function or information beyond aesthetics to users who can view the images.

Mark up purely decorative images so they can be ignored by assistive technology by using an empty `alt` attribute (`alt=""`). While it is not unacceptable to include decorative images inline, it is recommended if they are purely decorative to include the image using CSS.

EXAMPLE 374

Here's an example of an image being used as a decorative banner for a person's blog, the image offers no information and so an empty `alt` attribute is used.



Clara's Blog Welcome to my blog...

```
<header>
<div></div>
<h1>Clara's Blog</h1>
</header>
<p>Welcome to my blog...</p>
```

§ 4.7.5.1.10. INLINE IMAGES

When images are used inline as part of the flow of text in a sentence, provide a word or phrase as a text alternative which makes sense in the context of the sentence it is apart of.

EXAMPLE 375

I  you.

I `` you.

My  breaks.

My `` breaks.

§ 4.7.5.1.11. A GROUP OF IMAGES THAT FORM A SINGLE LARGER PICTURE WITH NO LINKS

When a picture has been sliced into smaller image files that are then displayed together to form the complete picture again, include a text alternative for one of the images using the `alt` attribute as per the relevant guidance for the picture as a whole, and then include an empty `alt` attribute on the other images.

EXAMPLE 376

In this example, a picture representing a company logo for the *PIP Corporation* has been split into two pieces, the first containing the letters "PIP" and the second with the word "CO". The text alternative *PIP CO* is in the `alt` attribute of the first image.



``

EXAMPLE 377

In the following example, a rating is shown as three filled stars and two empty stars. While the text alternative could have been "★★★☆☆", the author has instead decided to more helpfully give the rating in the form "3 out of 5". That is the text alternative of the first image, and the rest have empty `alt` attributes. 

```
<p>Rating:



</p>
```

§ 4.7.5.1.12. IMAGE MAPS

If an `<img>` element has a `usemap` attribute which references a `<map>` element containing `<area>` elements that have `href` at-

tributes, the `img` is considered to be interactive content. In such cases, always provide a text alternative for the image using the `alt` attribute.

EXAMPLE 378

Consider the following image which is a map of [Katoomba](#), it has 2 interactive regions corresponding to the areas of North and South Katoomba:



The text alternative is a brief description of the image. The `alt` attribute on each of the `<area>` elements provides text describing the content of the target page of each linked region:

```
<p>View houses for sale in North Katoomba or South Katoomba:</p>
<p>

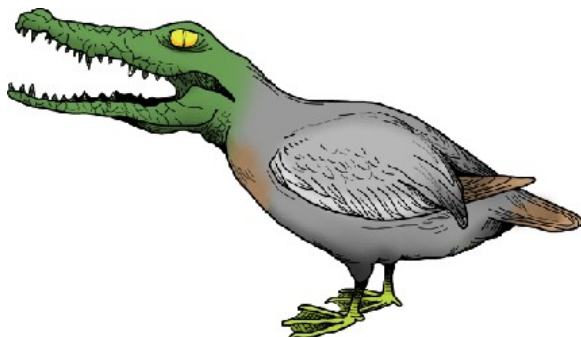
<map name="Map">
<area shape="poly" coords="78,124,124,10,189,29,173,93,168,132,136,151,110,130"
href="north.html" alt="Houses in North Katoomba">
<area shape="poly"
coords="66,63,80,135,106,138,137,154,167,137,175,133,144,240,49,223,17,137,17,61"
alt="Houses in South Katoomba" href="south.html">
</map>
```

4.7.5.1.13. A GROUP OF IMAGES THAT FORM A SINGLE LARGER PICTURE WITH LINKS

Sometimes, when you create a composite picture from multiple images, you may wish to link one or more of the images. Provide an `alt` attribute for each linked image to describe the purpose of the link.

EXAMPLE 379

In the following example, a composite picture is used to represent a "crocoduck"; a fictional creature which defies evolutionary principles by being part crocodile and part duck. You are asked to interact with the crocoduck, but you need to exercise caution...



```
<h1>The crocoduck</h1>
<p>You encounter a strange creature called a "crocoduck".
The creature seems angry! Perhaps some friendly stroking will help to calm
it, but be careful not to stroke any crocodile parts. This would just enrage
the beast further.</p>
<a href="?stroke=head"></a>
<a href="?stroke=body"></a>
```

§ 4.7.5.1.14. IMAGES OF PICTURES

Images of pictures or graphics include visual representations of objects, people, scenes, abstractions, etc. This [non-text content](#), [WCAG20] can convey a significant amount of information visually or provide a [specific sensory experience](#), [WCAG20] to a sighted person. Examples include photographs, paintings, drawings and artwork.

An appropriate text alternative for a picture is a brief description, or name [WCAG20]. As in all text alternative authoring decisions, writing suitable text alternatives for pictures requires human judgment. The text value is subjective to the context where the image is used and the page author's writing style. Therefore, there is no single "right" or "correct" piece of alt text for any particular image. In addition to providing a short text alternative that gives a brief description of the [non-text content](#), also providing supplemental content through another means when appropriate may be useful.

EXAMPLE 380

This first example shows an image uploaded to a photo-sharing site. The photo is of a cat, sitting in the bath. The image has a text alternative provided using the `<img>` element's `alt` attribute. It also has a caption provided by including the `<img>` element in a `<figure>` element and using a `<figcaption>` element to identify the caption text.



Lola prefers a bath to a shower.

```
<figure>
  
  <figcaption>Lola prefers a bath to a shower.</figcaption>
</figure>
```

EXAMPLE 381

This example is of an image that defies a complete description, as the subject of the image is open to interpretation. The image has a text alternative in the `alt` attribute which gives users who cannot view the image a sense of what the image is. It also has a caption provided by including the `<img>` element in a `<figure>` element and using a `<figcaption>` element to identify the caption text.



The first of the ten cards in the Rorschach test.

```
<figure>
  
  <figcaption>The first of the ten cards in the Rorschach test.</figcaption>
</figure>
```

§ 4.7.5.1.15. WEBCAM IMAGES

Webcam images are static images that are automatically updated periodically. Typically the images are from a fixed viewpoint, the images may update on the page automatically as each new image is uploaded from the camera or the user may be required to refresh the page to view an updated image. Examples include traffic and weather cameras.

EXAMPLE 382

This example is fairly typical; the title and a time stamp are included in the image, automatically generated by the webcam software. It would be better if the text information was not included in the image, but as it is part of the image, include it as part of the text alternative. A caption is also provided using the `<figure>` and `<figcaption>` elements. As the image is provided to give a visual indication of the current weather near a building, a link to a local weather forecast is provided, as with automatically generated and uploaded webcam images it may be impractical to provide such information as a text alternative.

The text of the `alt` attribute includes a prose version of the timestamp, designed to make the text more understandable when announced by text to speech software. The text alternative also includes a description of some aspects of what can be seen in the image which are unchanging, although weather conditions and time of day change.



View from the top of Sopwith house, looking towards North Kingston. This image is updated every hour.

View the [latest weather details](https://news.bbc.co.uk/weather/forecast/4296?area=Kingston) for Kingston upon Thames.

```
<figure>
  
  <figcaption>View from Sopwith house, looking towards north Kingston. This image is updated
  every hour.</figcaption>
</figure>
<p>View the <a href="https://news.bbc.co.uk/weather/forecast/4296?area=Kingston">latest
weather details</a> for Kingston upon Thames.</p>
```

§ 4.7.5.1.16. WHEN A TEXT ALTERNATIVE IS NOT AVAILABLE AT THE TIME OF PUBLICATION

In some cases an image is included in a published document, but the author is unable to provide an appropriate text alternative. In such cases the minimum requirement is to provide a caption for the image using the `figure` and `figcaption` elements under the following conditions:

- The `<img>` element is in a `<figure>` element
- The `<figure>` element contains a `<figcaption>` element
- The `<figcaption>` element contains content other than inter-element whitespace
- Ignoring the `<figcaption>` element and its descendants, the `figure` element has no Text node descendants other than inter-element whitespace, and no embedded content descendant other than the `<img>` element.

NOTE:

In other words, the only content of the `figure` is an `<img>` element and a `figcaption` element, and the `<figcaption>` element must include (caption) content.

NOTE:

Such cases are to be kept to an absolute minimum. If there is even the slightest possibility of the author having the ability to provide real alternative text, then it would not be acceptable to omit the `alt` attribute.

EXAMPLE 383

In this example, a person uploads a photo, as part of a bulk upload of many images, to a photo sharing site. The user has not provided a text alternative or a caption for the image. The site's authoring tool inserts a caption automatically using whatever useful information it has for the image. In this case it's the file name and date the photo was taken.

⚠Warning! The caption text in the example below is not a suitable text alternative and is not conforming to the Web Accessibility Guidelines 2.0. [\[WCAG20\]](#)



clara.jpg, taken on 12/11/2010.

```
<figure>

<figcaption>clara.jpg, taken on 12/11/2010.</figcaption>
</figure>
```

Notice that even in this example, as much useful information as possible is still included in the `<figcaption>` element.

EXAMPLE 384

In this second example, a person uploads a photo to a photo sharing site. She has provided a caption for the image but not a text alternative. This may be because the site does not provide users with the ability to add a text alternative in the `alt` attribute.



Eloisa with Princess Belle

```
<figure>

<figcaption>Eloisa with Princess Belle</figcaption>
</figure>
```

EXAMPLE 385

Sometimes the entire point of the image is that a textual description is not available, and the user is to provide the description. For example, software that displays images and asks for alternative text precisely for the purpose of then writing a page with correct alternative text. Such a page could have a table of images, like this:

```
<table>
  <tr><tr> <th> Image <th> Description<tr>
  <td>
    <figure>
      
      <figcaption>Image 640 by 100, filename 'banner.gif'</figcaption>
    </figure>
    <td> <input name="alt2421">
  <tr>
    <td> <figure>
      
      <figcaption>Image 200 by 480, filename 'ad3.gif'</figcaption>
    </figure>
    <td> <input name="alt2422">
  </table>
```

NOTE:

Since some users cannot use images at all (e.g., because they are blind) the `alt` attribute is only allowed to be omitted when no text alternative is available and none can be made available, as in the above examples.

§ 4.7.5.1.17. AN IMAGE NOT INTENDED FOR THE USER

Generally authors should avoid using `<img>` elements for purposes other than showing images.

If an `<img>` element is being used for purposes other than showing an image, e.g., as part of a service to count page views, use an empty `alt` attribute.

EXAMPLE 386

An example of an `<img>` element used to collect web page statistics. The `alt` attribute is empty as the image has no meaning.

```

```

NOTE:

It is recommended for the example use above the `width` and `height` attributes be set to zero.

EXAMPLE 387

Another example use is when an image such as a *spacer.gif* is used to aid positioning of content. The `alt` attribute is empty as the image has no meaning.

```

```

NOTE:

It is recommended that CSS be used to position content instead of `<img>` elements.

§ 4.7.5.1.18. ICON IMAGES

An icon is usually a simple picture representing a program, action, data file or a concept. Icons are intended to help users of visual browsers to recognize features at a glance.

Use an empty `alt` attribute when an icon is supplemental to text conveying the same meaning.

EXAMPLE 388

In this example, we have a link pointing to a site's home page, the link contains a house icon image and the text "home". The image has an empty `alt` text.



```
<a href="home.html">Home</a>
```

Where images are used in this way, it would also be appropriate to add the image using CSS.

```
#home:before
{
  content: url(home.png);
}

<a href="home.html" id="home">Home</a>
```

EXAMPLE 389

In this example, there is a warning message, with a warning icon. The word "Warning!" is in emphasized text next to the icon. As the information conveyed by the icon is redundant the `<img>` element is given an empty `alt` attribute.



Warning! Your session is about to expire.

```
<p>
<strong>Warning!</strong>
Your session is about to expire</p>
```

When an icon conveys additional information not available in text, provide a text alternative.

EXAMPLE 390

In this example, there is a warning message, with a warning icon. The icon emphasizes the importance of the message and identifies it as a particular type of content.



Your session is about to expire.

```
<p>
Your session is about to expire</p>
```

§ 4.7.5.1.19. LOGOS, INSIGNIA, FLAGS, OR EMBLEMS

Many pages include logos, insignia, flags, or emblems, which stand for a company, organization, project, band, software package, country, or other entity. What can be considered as an appropriate text alternative depends upon, like all images, the context in which the image is being used and what function it serves in the given context.

If a logo is the sole content of a link, provide a brief description of the link target in the alt attribute.

EXAMPLE 391

This example illustrates the use of the HTML5 logo as the sole content of a link to the HTML specification.



```
<a href="https://w3c.github.io/html/">
</a>
```

If a logo is being used to represent the entity, e.g., as a page heading, provide the name of the entity being represented by the logo as the text alternative.

EXAMPLE 392

This example illustrates the use of the WebPlatform.org logo being used to represent itself.



WebPlatform.org and other developer resources

```
<h2> and other developer resources</h2>
```

NOTE:

The text alternative in the example above could also include the word "logo" to describe the type of image content. If so, it is suggested that square brackets be used to delineate this information: alt="[logo] WebPlatform.org".

If a logo is being used next to the name of the what that it represents, then the logo is supplemental. Include an empty alt attribute as the text alternative is already provided.

EXAMPLE 393

This example illustrates the use of a logo next to the name of the organization it represents.



WebPlatform.org

```
 WebPlatform.org
```

If the logo is used alongside text discussing the subject or entity the logo represents, then provide a text alternative which describes the logo.

EXAMPLE 394

This example illustrates the use of a logo next to text discussing the subject the logo represents.

HTML is a language for structuring and presenting content for the World Wide Web, a core technology of the Internet. It is the latest revision of the HTML standard (originally created in 1990 and most recently standardized as HTML 4.01 in 1997) and currently remains under development. Its core aims have been to improve the language with support for the latest multimedia while keeping it easily readable by humans and consistently understood by computers and devices (web browsers, parsers etc.).

```
<p></p>
```

Information about HTML

§ 4.7.5.1.20. CAPTCHA IMAGES

CAPTCHA stands for "Completely Automated Public Turing test to tell Computers and Humans Apart". CAPTCHA images are used for security purposes to confirm that content is being accessed by a person rather than a computer. This authentication is done through visual verification of an image. CAPTCHA typically presents an image with characters or words in it that the user is to re-type. The image is usually distorted and has some noise applied to it to make the characters difficult to read.

To improve the accessibility of CAPTCHA provide text alternatives that identify and describe the purpose of the image, and provide alternative forms of the CAPTCHA using output modes for different types of sensory perception. For instance provide an audio alternative along with the visual image. Place the audio option right next to the visual one. This helps but is still problematic for people without sound cards, the deaf-blind, and some people with limited hearing. Another method is to include a form that asks a question along with the visual image. This helps but can be problematic for people with cognitive impairments.

NOTE:

It is strongly recommended that alternatives to CAPTCHA be used, as all forms of CAPTCHA introduce unacceptable barriers to entry for users with disabilities. Further information is available in [Inaccessibility of CAPTCHA](#).

EXAMPLE 395

This example shows a CAPTCHA test which uses a distorted image of text. The text alternative in the `alt` attribute provides instructions for a user in the case where she cannot access the image content.



Example code:

```

<!-- audio CAPTCHA option that allows the user to listen and type the word -->
<!-- form that asks a question -->
```

§ 4.7.5.1.21. AN IMAGE IN A `<picture>` ELEMENT

The `<picture>` element and any `<source>` elements it contains have no semantics for users, only the `<img>` element or its text alternative is displayed to users. Provide a text alternative for an `<img>` element without regard to it being within a `<picture>` element. Refer to [Requirements for providing text to act as an alternative for images](#) for more information on how to provide useful `alt` text for images.

NOTE:

[Art directed](#) images that rely on picture need to depict the same content (irrespective of size, pixel density, or any other discriminating factor). Therefore the appropriate text alternative for an image will always be the same irrespective of which source file ends up being chosen by the browser.

EXAMPLE 396

```
<h2>Is it a ghost?</h2>
<picture>
  <source media="(min-width: 32em)" srcset="large.jpg">
  
</picture>
```

The large and small versions (both versions are displayed for demonstration purposes) of the image portray the same scene: Reflection of a girls face in a train window, while the small version (displayed on smaller screens) is cropped, this does not effect the subject matter or the appropriateness of the `alt` text.



§ 4.7.5.1.22. GUIDANCE FOR MARKUP GENERATORS

Markup generators (such as WYSIWYG authoring tools) should, wherever possible, obtain alternative text from their users.

However, it is recognized that in many cases, this will not be possible.

For images that are the sole contents of links, markup generators should examine the link target to determine the title of the target, or the URL of the target, and use information obtained in this manner as the alternative text.

For images that have captions, markup generators should use the [<figure>](#) and [<figcaption>](#) elements to provide the image's caption.

As a last resort, implementors should either set the `alt` attribute to the empty string, under the assumption that the image is a purely decorative image that doesn't add any information but is still specific to the surrounding content, or omit the `alt` attribute altogether, under the assumption that the image is a key part of the content.

Markup generators may specify a **generator-unable-to-provide-required-alt** attribute on [](#) elements for which they have been unable to obtain a text alternative and for which they have therefore omitted the `alt` attribute. The value of this attribute must be the empty string. Documents containing such attributes are not conforming, but conformance checkers will silently ignore this error.

NOTE:

This is intended to avoid markup generators from being pressured into replacing the error of omitting the `alt` attribute with the even more egregious error of providing phony text alternatives, because state-of-the-art automated conformance checkers cannot distinguish phony text alternatives from correct text alternatives.

Markup generators should generally avoid using the image's own file name as the text alternative. Similarly, markup generators should avoid generating text alternatives from any content that will be equally available to presentation user agents (e.g., Web browsers).

NOTE:

This is because once a page is generated, it will typically not be updated, whereas the browsers that later read the page can be updated by the user, therefore the browser is likely to have more up-to-date and finely-tuned heuristics than the markup generator did when generating the page.

§ 4.7.5.1.23. GUIDANCE FOR CONFORMANCE CHECKERS

A conformance checker must report the lack of an `alt` attribute as an error unless one of the conditions listed below applies:

- The [](#) element is in a [<figure>](#) element that satisfies [the conditions described above](#).
- The [](#) element has a (non-conforming) `generator-unable-to-provide-required-alt` attribute whose value is the empty string. A conformance checker that is not reporting the lack of an `alt` attribute as an error must also not report the presence of the empty `generator-unable-to-provide-required-alt` attribute as an error. (This case does not represent a case where the document is conforming, only that the generator could not determine appropriate alternative text — validators are not required to show an error in this case, because such an error might encourage markup generators to include bogus alternative text purely in an attempt to silence validators. Naturally, conformance checkers *may* report the lack of an `alt` attribute as an error even in the presence of the `generator-unable-to-provide-required-alt` attribute; for example, there could be a user option to report *all* conformance errors even those that might be the more or less inevitable result of using a markup generator.)

§ 4.7.6. The `iframe` element

Categories:

[Flow content.](#)
[Phrasing content.](#)
[Embedded content.](#)
[Interactive content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [embedded content](#) is expected.

Content model:

Text that conforms to the requirements given in the prose.

Tag omission in text/html:

Neither tag is omissible

Content attributes:Global attributes

`src` - Address of the resource

`srcdoc` - A document to render in the `iframe`

`name` - Name of nested browsing context

`sandbox` - Security rules for nested content

`allowfullscreen` - Whether to allow the `iframe`'s contents to use `requestFullscreen()`

`width` - Horizontal dimension

`height` - Vertical dimension

Allowed ARIA role attribute values:

`'application'`, `'document'`, or `'img'`.

Allowed ARIA state and property attributes:Global aria-* attributes

Any aria-* attributes applicable to the allowed roles.

DOM interface:

```
interface HTMLIFrameElement : HTMLElement {
  attribute DOMString src;
  attribute DOMString srcdoc;
  attribute DOMString name;
  [PutForwards=value] readonly attribute DOMTokenList sandbox;
  attribute boolean allowFullscreen;
  attribute DOMString width;
  attribute DOMString height;
  readonly attribute Document? contentDocument;
  readonly attribute WindowProxy? contentWindow;
};
```

The `<iframe>` element represents a nested browsing context.

The `src` attribute gives the address of a page that the nested browsing context is to contain. The attribute, if present, must be a valid non-empty URL potentially surrounded by spaces.

The `srcdoc` attribute gives the content of the page that the nested browsing context is to contain. The value of the attribute is the source of an **iframe srcdoc document**.

The `srcdoc` attribute, if present, must have a value using the HTML syntax that consists of the following syntactic components, in the given order:

1. Any number of comments and space characters.
2. Optionally, a DOCTYPE.
3. Any number of comments and space characters.
4. The root element, in the form of an `<html>` element.
5. Any number of comments and space characters.

For `<iframe>` elements in XML documents, the `srcdoc` attribute, if present, must have a value that matches the production labeled document in the XML specification. [XML]

EXAMPLE 397

Here a blog uses the `srcdoc` attribute in conjunction with the [sandbox](#) attributes described below to provide users of user agents that support this feature with an extra layer of protection from script injection in the blog post comments:

```
<article>
  <h1>I got my own magazine!</h1>
  <p>After much effort, I've finally found a publisher, and so now I
  have my own magazine! Isn't that awesome?! The first issue will come
  out in September, and we have articles about getting food, and about
  getting in boxes, it's going to be great!</p>
  <footer>
    <p>Written by <a href="/users/cap">cap</a>, 1 hour ago.
  </footer>
  <article>
    <footer> Thirteen minutes ago, <a href="/users/ch">ch</a> wrote: </footer>
    <iframe sandbox srcdoc="<p>did you get a cover picture yet?"></iframe>
  </article>
  <article>
    <footer> Nine minutes ago, <a href="/users/cap">cap</a> wrote: </footer>
    <iframe sandbox srcdoc="<p>Yeah, you can see it <a href=&quot;/gallery?mode=cover&amp;
page=1&quot;>in my gallery</a>."></iframe>
  </article>
  <article>
    <footer> Five minutes ago, <a href="/users/ch">ch</a> wrote: </footer>
    <iframe sandbox srcdoc="<p>hey that's earl's table.
<p>you should get earl&amp;me on the next cover."></iframe>
  </article>
```

Notice the way that quotes have to be escaped (otherwise the `srcdoc` attribute would end prematurely), and the way raw ampersands (e.g., in URLs or in prose) mentioned in the sandboxed content have to be *doubly* escaped — once so that the ampersand is preserved when originally parsing the `srcdoc` attribute, and once more to prevent the ampersand from being misinterpreted when parsing the sandboxed content.

Furthermore, notice that since the [DOCTYPE](#) is optional in [iframe srcdoc documents](#), and the `<html>`, `<head>`, and `<body>` elements have [optional start and end tags](#), and the `<title>` element is also optional in [iframe srcdoc documents](#), the markup in a `srcdoc` attribute can be relatively succinct despite representing an entire document, since only the contents of the `<body>` element need appear literally in the syntax. The other elements are still present, but only by implication.

NOTE:

In [the HTML syntax](#), authors need only remember to use U+0022 QUOTATION MARK characters (") to wrap the attribute contents and then to escape all U+0022 QUOTATION MARK (") and U+0026 AMPERSAND (&) characters, and to specify the [sandbox](#) attribute, to ensure safe embedding of content.

NOTE:

Due to restrictions of [the XHTML syntax](#), in XML the U+003C LESS-THAN SIGN character (<) needs to be escaped as well. In order to prevent [attribute-value normalization](#), some of XML's whitespace characters — specifically U+0009 CHARACTER TABULATION (tab), U+000A LINE FEED (LF), and U+000D CARRIAGE RETURN (CR) — also need to be escaped. [\[XML\]](#)

NOTE:

If the `src` attribute and the `srcdoc` attribute are both specified together, the `srcdoc` attribute takes priority. This allows authors to provide a fallback [URL](#) for legacy user agents that do not support the `srcdoc` attribute.



When an `<iframe>` element is [inserted into a document](#) that has a [browsing context](#), the user agent must create a [nested](#)

[browsing context](#), and then [process the iframe attributes](#) for the "first time".

When an `<iframe>` element is [removed from a document](#), the user agent must [discard](#) the [nested browsing context](#), if any.

NOTE:

This happens without any unload events firing (the [nested browsing context](#) and its [Document](#) are [discarded](#), not [unloaded](#)).

Whenever an `<iframe>` element with a [nested browsing context](#) has its `srcdoc` attribute set, changed, or removed, the user agent must [process the iframe attributes](#).

Similarly, whenever an `<iframe>` element with a [nested browsing context](#) but with no `srcdoc` attribute specified has its `src` attribute set, changed, or removed, the user agent must [process the iframe attributes](#).

When the user agent is to **process the iframe attributes**, it must run the first appropriate steps from the following list:

↪ **If the `srcdoc` attribute is specified**

[Navigate](#) the element's [child browsing context](#) to a new [response](#) whose [url list](#) consists of [about:srcdoc](#), [header list](#) consists of Content-Type/`text/html`, [body](#) is the value of the attribute, [CSP list](#) is the [CSP list](#) of the `<iframe>` element's [node document](#), and [HTTPS state](#) is the [HTTPS state](#) of the `<iframe>` element's [node document](#).

The resulting [Document](#) must be considered [an iframe srcdoc document](#).

↪ **Otherwise, if the element has no `src` attribute specified, and the user agent is processing the iframe's attributes for the "first time"**

[Queue a task](#) to run the [iframe load event steps](#).

The [task source](#) for this [task](#) is the [DOM manipulation task source](#).

↪ **Otherwise**

1. If the element has no `src` attribute specified, or its value is the empty string, let *url* be the string `"about:blank"`.

Otherwise, [parse](#) the value of the `src` attribute, relative to the `<iframe>` element.

If that is not successful, then let *url* be the string `"about:blank"`. Otherwise, let *url* be the [resulting URL string](#).

2. If there exists an [ancestor browsing context](#) whose [active document](#)'s [address](#), ignoring fragment identifiers, is equal to *url*, then abort these steps.
3. [Navigate](#) the element's [child browsing context](#) to *url*.

Furthermore, if the [active document](#) of the element's [child browsing context](#) before such a [navigation](#) was not [completely loaded](#) at the time of the new [navigation](#), then the [navigation](#) must be completed with [replacement enabled](#).

Similarly, if the [child browsing context](#)'s [session history](#) contained only one [Document](#) when the [process the iframe attributes](#) algorithm was invoked, and that was the `about:blank` [Document](#) created when the [child browsing context](#) was created, then any [navigation](#) required of the user agent in that algorithm must be completed with [replacement enabled](#).

When a [Document](#) in an `iframe` is marked as [completely loaded](#), the user agent must run the [iframe load event steps in parallel](#).

NOTE:

A load event is also fired at the `<iframe>` element when it is created if no other data is loaded in it.

Each [Document](#) has an **iframe load in progress** flag and a **mute iframe load** flag. When a [Document](#) is created, these flags must be unset for that [Document](#).

The **iframe load event steps** are as follows:

1. Let *child document* be the [active document](#) of the `<iframe>` element's [nested browsing context](#).

2. If *child document* has its [mute iframe load](#) flag set, abort these steps.
3. Set *child document*'s [iframe load in progress](#) flag.
4. [Fire a simple event](#) named load at the `<iframe>` element.
5. Unset *child document*'s [iframe load in progress](#) flag.

⚠Warning! This, in conjunction with scripting, can be used to probe the URL space of the local network's HTTP servers. User agents may implement [cross-origin](#) access control policies that are stricter than those described above to mitigate this attack, but unfortunately such policies are typically not compatible with existing Web content.

When the *iframe*'s [browsing context](#)'s [active document](#) is not [ready for post-load tasks](#), and when anything in the *iframe* is [delaying the load event](#) of the *iframe*'s [browsing context](#)'s [active document](#), and when the *iframe*'s [browsing context](#) is in the [delaying load events mode](#), the *iframe* must [delay the load event](#) of its document.

NOTE:

If, during the handling of the load event, the [browsing context](#) in the *iframe* is again [navigated](#), that will further [delay the load event](#).

NOTE:

If, when the element is created, the `srcdoc` attribute is not set, and the `src` attribute is either also not set or set but its value cannot be [resolved](#), the *browsing context* will remain at the initial `about:blank` page.

NOTE:

If the user [navigates](#) away from this page, the *iframe*'s corresponding `WindowProxy` object will proxy new `Window` objects for new [Document](#) objects, but the `src` attribute will not change.



The `name` attribute, if present, must be a [valid browsing context name](#). The given value is used to name the [nested browsing context](#). When the *browsing context* is created, if the attribute is present, the [browsing context name](#) must be set to the value of this attribute; otherwise, the [browsing context name](#) must be set to the empty string.

Whenever the `name` attribute is set, the nested *browsing context*'s [name](#) must be changed to the new value. If the attribute is removed, the [browsing context name](#) must be set to the empty string.



The `sandbox` attribute, when specified, enables a set of extra restrictions on any content hosted by the `<iframe>`. Its value must be an [unordered set of unique space-separated tokens](#) that are [ASCII case-insensitive](#). The allowed values are `allow-forms`, `allow-pointer-lock`, `allow-popups`, `allow-same-origin`, `allow-scripts`, and `allow-top-navigation`.

When the attribute is set, the content is treated as being from a unique [origin](#), forms, scripts, and various potentially annoying APIs are disabled, links are prevented from targeting other [browsing contexts](#), and plugins are secured. The `allow-same-origin` keyword causes the content to be treated as being from its real origin instead of forcing it into a unique origin; the `allow-top-navigation` keyword allows the content to [navigate its top-level browsing context](#); and the `allow-forms`, `allow-pointer-lock`, `allow-popups` and `allow-scripts` keywords re-enable forms, the pointer lock API, popups, and scripts respectively. [\[POINTERLOCK\]](#)

⚠Warning! Setting both the `allow-scripts` and `allow-same-origin` keywords together when the embedded page has the [same origin](#) as the page containing the *iframe* allows the embedded page to simply remove the [sandbox](#) attribute and then reload itself, effectively breaking out of the sandbox altogether.

⚠Warning! These flags only take effect when the [nested browsing context](#) of the `iframe` is [navigated](#). Removing them, or removing the entire [sandbox](#) attribute, has no effect on an already-loaded page.

⚠Warning! Potentially hostile files should not be served from the same server as the file containing the `<iframe>` element. Sandboxing hostile content is of minimal help if an attacker can convince the user to just visit the hostile content directly, rather than in the `<iframe>`. To limit the damage that can be caused by hostile HTML content, it should be served from a separate dedicated domain. Using a different domain ensures that scripts in the files are unable to attack the site, even if the user is tricked into visiting those pages directly, without the protection of the [sandbox](#) attribute.

When an `<iframe>` element with a [sandbox](#) attribute has its [nested browsing context](#) created (before the initial `about:blank Document` is created), and when an `iframe` element's [sandbox](#) attribute is set or changed while it has a [nested browsing context](#), the user agent must [parse the sandboxing directive](#) using the attribute's value as the *input*, the `<iframe>` element's [nested browsing context](#)'s `<iframe>` sandboxing flag set as the output, and, if the `iframe` has an `allowfullscreen` attribute, the *allow fullscreen flag*.

When an `<iframe>` element's [sandbox](#) attribute is removed while it has a [nested browsing context](#), the user agent must empty the `<iframe>` element's [nested browsing context](#)'s `<iframe>` sandboxing flag set as the output.

EXAMPLE 398

In this example, some completely-unknown, potentially hostile, user-provided HTML content is embedded in a page. Because it is served from a separate domain, it is affected by all the normal cross-site restrictions. In addition, the embedded page has scripting disabled, plugins disabled, forms disabled, and it cannot navigate any frames or windows other than itself (or any frames or windows it itself embeds).

```
<p>We're not scared of you! Here is your content, unedited:</p>
<iframe title="Example iframe" sandbox src="https://usercontent.example.net
/getusercontent.cgi?id=12193"></iframe>
```

⚠Warning! It is important to use a separate domain so that if the attacker convinces the user to visit that page directly, the page doesn't run in the context of the site's origin, which would make the user vulnerable to any attack found in the page.

EXAMPLE 399

In this example, a gadget from another site is embedded. The gadget has scripting and forms enabled, and the origin sandbox restrictions are lifted, allowing the gadget to communicate with its originating server. The sandbox is still useful, however, as it disables plugins and popups, thus reducing the risk of the user being exposed to malware and other annoyances.

```
<iframe title="Maps" sandbox="allow-same-origin allow-forms allow-scripts"
src="https://maps.example.com/embedded.html"></iframe>
```

EXAMPLE 400

Suppose a file A contained the following fragment:

```
<iframe title="Example iframe" sandbox="allow-same-origin allow-forms" src=B></iframe>
```

Suppose that file B contained an iframe also:

```
<iframe title="Example iframe" sandbox="allow-scripts" src=C></iframe>
```

Further, suppose that file C contained a link:

```
<a href=D>Link</a>
```

For this example, suppose all the files were served as [text/html](#).

Page C in this scenario has all the sandboxing flags set. Scripts are disabled, because the `iframe` in A has scripts disabled, and this overrides the `allow-scripts` keyword set on the `iframe` in B. Forms are also disabled, because the inner `iframe` (in B) does not have the `allow-forms` keyword set.

Suppose now that a script in A removes all the [sandbox](#) attributes in A and B. This would change nothing immediately. If the user clicked the link in C, loading page D into the `iframe` in B, page D would now act as if the `iframe` in B had the `allow-same-origin` and `allow-forms` keywords set, because that was the state of the [nested browsing context](#) in the `iframe` in A when page B was loaded.

Generally speaking, dynamically removing or changing the [sandbox](#) attribute is ill-advised, because it can make it quite hard to reason about what will be allowed and what will not.



The `allowfullscreen` attribute is a [boolean attribute](#). When specified, it indicates that [Document](#) objects in the `<iframe>` element's [browsing context](#) are to be allowed to use `requestFullscreen()` (if it's not blocked for other reasons, e.g., there is another ancestor `iframe` without this attribute set).

EXAMPLE 401

Here, an `iframe` is used to embed a player from a video site. The `allowfullscreen` attribute is needed to enable the player to show its video fullscreen.

```
<article>
  <header>
    <p> <b>Fred Flintstone</b></p>
    <p><a href="/posts/3095182851" rel=bookmark>12:44</a> – <a href="#acl-3095182851">Private
  Post</a></p>
  </header>
  <main>
    <p>Check out my new ride!</p>
    <iframe title="Video" src="https://video.example.com/embed?id=92469812" allowfullscreen>
  </iframe>
  </main>
</article>
```



The `<iframe>` element supports [dimension attributes](#) for cases where the embedded content has specific dimensions (e.g., ad units have well-defined dimensions).

An `<iframe>` element never has [fallback content](#), as it will always create a nested [browsing context](#), regardless of whether

the specified initial contents are successfully used.



Descendants of `<iframe>` elements represent nothing. (In legacy user agents that do not support `<iframe>` elements, the contents would be parsed as markup that could act as fallback content.)

When used in [HTML documents](#), the allowed content model of `<iframe>` elements is text, except that invoking the [HTML fragment parsing algorithm](#) with the `<iframe>` element as the [context](#) element and the text contents as the *input* must result in a list of nodes that are all [phrasing content](#), with no [parse errors](#) having occurred, with no `script` elements being anywhere in the list or as descendants of elements in the list, and with all the elements in the list (including their descendants) being themselves conforming.

The `<iframe>` element must be empty in [XML documents](#).

NOTE:

The [HTML parser](#) treats markup inside `<iframe>` elements as text.



The IDL attributes `src`, `srcdoc`, `name`, and `sandbox` must [reflect](#) the respective content attributes of the same name.

The [supported tokens](#) for `sandbox`'s [DOMTokenList](#) are the allowed values defined in the `sandbox` attribute and supported by the user agent.

The `allowFullscreen` IDL attribute must [reflect](#) the `allowFullscreen` content attribute.

The `contentDocument` IDL attribute must return the [Document](#) object of the [active document](#) of the `<iframe>` element's [nested browsing context](#), if any and if its [origin](#) is the [same origin-domain](#) as the [origin](#) specified by the [incumbent settings object](#), or null otherwise.

The `contentWindow` IDL attribute must return the [WindowProxy](#) object of the `<iframe>` element's [nested browsing context](#), if any, or null otherwise.

EXAMPLE 402

Here is an example of a page using an `iframe` to include advertising from an advertising broker:

```
<iframe title="Advert" src="https://ads.example.com/?customerid=923513721&format=banner"
width="468" height="60"></iframe>
```

4.7.7. The embed element

Categories:

- [Flow content.](#)
- [Phrasing content.](#)
- [Embedded content.](#)
- [Interactive content.](#)
- [Palpable content.](#)

Contexts in which this element can be used:

Where [embedded content](#) is expected.

Content model:

[Nothing.](#)

Tag omission in text/html:

No [end tag](#)

Content attributes:

[Global attributes](#)

`src` - Address of the resource

`type` - Type of embedded resource

`width` - Horizontal dimension

`height` - Vertical dimension

Any other attribute that has no namespace (see prose).

Allowed [ARIA role attribute](#) values:

[‘application’](#), [‘document’](#) or [‘img’](#) or [‘presentation’](#).

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLEmbedElement : HTMLElement {
  attribute DOMString src;
  attribute DOMString type;
  attribute DOMString width;
  attribute DOMString height;
  legacycaller any (any... arguments);
};
```

Depending on the type of content instantiated by the `<embed>` element, the node may also support other interfaces.

The `<embed>` element provides an integration point for an external (typically non-HTML) application or interactive content.

The `src` attribute gives the address of the resource being embedded. The attribute, if present, must contain a [valid non-empty URL potentially surrounded by spaces](#).

The `type` attribute, if present, gives the [MIME type](#) by which the plugin to instantiate is selected. The value must be a [valid mime type](#). If both the `type` attribute and the `src` attribute are present, then the `type` attribute must specify the same type as the [explicit Content-Type metadata](#) of the resource given by the `src` attribute.

While any of the following conditions are occurring, any [plugin](#) instantiated for the element must be removed, and the `<embed>` element [represents](#) nothing:

- The element has neither a `src` attribute nor a `type` attribute.
- The element has a [media element](#) ancestor.
- The element has an ancestor `<object>` element that is *not* showing its [fallback content](#).

An `<embed>` element is said to be **potentially active** when the following conditions are all met simultaneously:

- The element is [in a Document](#) or was [in a Document](#) the last time the [event loop](#) reached step 1.
- The element’s [node document](#) is [fully active](#).
- The element has either a `src` attribute set or a `type` attribute set (or both).
- The element’s `src` attribute is either absent or its value is not the empty string.
- The element is not a descendant of a [media element](#).
- The element is not a descendant of an `<object>` element that is not showing its [fallback content](#).
- The element is [being rendered](#), or was [being rendered](#) the last time the [event loop](#) reached step 1.

Whenever an `<embed>` element that was not [potentially active](#) becomes [potentially active](#), and whenever a [potentially active <embed>](#) element that is remaining [potentially active](#) and has its `src` attribute set, changed, or removed or its `type` attribute set, changed, or removed, the user agent must [queue a task](#) using the **embed task source** to run [the embed element setup steps](#).

The **`<embed>` element setup steps** are as follows:

1. If another [task](#) has since been queued to run [the embed element setup steps](#) for this element, then abort these steps.

2. **↻ If the element has a `src` attribute set**

The user agent must [parse](#) the value of the element’s `src` attribute, relative to the element. If that is successful, the user agent should run these steps:

1. Let `request` be a new [request](#) whose [URL](#) is the [resulting URL string](#), [client](#) is the element’s [node docu-](#)

ment's Window object's [environment settings object](#), [destination](#) is "unknown", [omit-Origin-header flag](#) is set if the element doesn't have a [browsing context scope origin](#), [credentials mode](#) is "include", and whose [use-URL-credentials flag](#) is set.

2. [Fetch request](#).

The [task](#) that is [queued](#) by the [networking task source](#) once the resource has been fetched must run the following steps:

1. If another [task](#) has since been queued to run [the embed element setup steps](#) for this element, then abort these steps.
2. Determine the **type of the content** being embedded, as follows (stopping at the first substep that determines the type):
 1. If the element has a [type](#) attribute, and that attribute's value is a type that a [plugin](#) supports, then the value of the [type](#) attribute is the content's type.
 2. Otherwise, if applying the [URL parser](#) algorithm to the [URL](#) of the specified resource (after any redirects) results in a [URL record](#) whose [path](#) component matches a pattern that a [plugin](#) supports, then the content's type is the type that the plugin can handle.

EXAMPLE 403

For example, a plugin might say that it can handle resources with [path](#) components that end with the four character string ".swf".

3. Otherwise, if the specified resource has [explicit Content-Type metadata](#), then that is the content's type.
4. Otherwise, the content has no [type](#) and there can be no appropriate [plugin](#) for it.
3. If the previous step determined that the content's type is `image/svg+xml`, then run the following substeps:
 1. If the `<embed>` element is not associated with a [nested browsing context](#), associate the element with a newly created [nested browsing context](#), and, if the element has a [name](#) attribute, set the [browsing context name](#) of the element's [nested browsing context](#) to the value of this attribute.
 2. [Navigate](#) the [nested browsing context](#) to the fetched resource, with [replacement enabled](#), and with the `<embed>` element's [node document](#)'s [browsing context](#) as the [source browsing context](#). (The `src` attribute of the `<embed>` element doesn't get updated if the browsing context gets further navigated to other locations.)
 3. The `<embed>` element now [represents](#) its associated [nested browsing context](#).
4. Otherwise, find and instantiate an appropriate [plugin](#) based on the content's type, and hand that [plugin](#) the content of the resource, replacing any previously instantiated plugin for the element. The `<embed>` element now represents this [plugin](#) instance.
5. Once the resource or plugin has completely loaded, [queue a task](#) to [fire a simple event](#) named `load` at the element.

Whether the resource is fetched successfully or not (e.g., whether the response status was an [ok status](#)) must be ignored when determining the content's type and when handing the resource to the plugin.

NOTE:

This allows servers to return data for plugins even with error responses (e.g., HTTP 500 Internal Server Error codes can still contain plugin data).

Fetching the resource must [delay the load event](#) of the element's [node document](#).

↪ If the element has no `src` attribute set

The user agent should find and instantiate an appropriate [plugin](#) based on the value of the [type](#) attribute. The

embed element now represents this [plugin](#) instance.

Once the plugin is completely loaded, [queue a task](#) to [fire a simple event](#) named load at the element.

The `<embed>` element has no [fallback content](#). If the user agent can't find a suitable plugin when attempting to find and instantiate one for the algorithm above, then the user agent must use a default plugin. This default could be as simple as saying "Unsupported Format".

Whenever an `<embed>` element that was [potentially active](#) stops being [potentially active](#), any [plugin](#) that had been instantiated for that element must be unloaded.

When a [plugin](#) is to be instantiated but it cannot be [secured](#) and the [sandboxed plugins browsing context flag](#) is set on the `<embed>` element's [node document](#)'s [active sandboxing flag set](#), then the user agent must not instantiate the [plugin](#), and must instead render the `<embed>` element in a manner that conveys that the [plugin](#) was disabled. The user agent may offer the user the option to override the sandbox and instantiate the [plugin](#) anyway; if the user invokes such an option, the user agent must act as if the conditions above did not apply for the purposes of this element.

⚠Warning! Plugins that cannot be [secured](#) are disabled in sandboxed browsing contexts because they might not honor the restrictions imposed by the sandbox (e.g., they might allow scripting even when scripting in the sandbox is disabled). User agents should convey the danger of overriding the sandbox to the user if an option to do so is provided.

When an `<embed>` element represents a [nested browsing context](#): if the `<embed>` element's [nested browsing context](#)'s [active document](#) is not [ready for post-load tasks](#), and when anything is [delaying the load event](#) of the `<embed>` element's [browsing context](#)'s [active document](#), and when the `<embed>` element's [browsing context](#) is in the [delaying load events mode](#), the embed must [delay the load event](#) of its document.

The [task source](#) for the [tasks](#) mentioned in this section is the [DOM manipulation task source](#).

Any namespace-less attribute other than name, align, hspace, and vspace may be specified on the `<embed>` element, so long as its name is [XML-compatible](#) and contains no [uppercase ASCII letters](#). These attributes are then passed as parameters to the [plugin](#).

NOTE:

All attributes in [HTML documents](#) get lowercased automatically, so the restriction on uppercase letters doesn't affect such documents.

NOTE:

The four exceptions are to exclude legacy attributes that have side-effects beyond just sending parameters to the [plugin](#).

The user agent should pass the names and values of all the attributes of the embed element that have no namespace to the [plugin](#) used, when one is instantiated.

The HTMLEmbedElement object representing the element must expose the scriptable interface of the [plugin](#) instantiated for the `<embed>` element, if any. At a minimum, this interface must implement the **legacy caller operation**. (It is suggested that the default behavior of this legacy caller operation, e.g., the behavior of the default plugin's legacy caller operation, be to throw a [NotSupportedError](#) exception.)

The `<embed>` element supports [dimension attributes](#).

The IDL attributes `src` and `type` each must [reflect](#) the respective content attributes of the same name.

EXAMPLE 404

Here's a way to embed a resource that requires a proprietary plugin, like Flash:

```
<embed src="catgame.swf">
```

If the user does not have the plugin (for example if the plugin vendor doesn't support the user's platform), then the user will be unable to use the resource.

To pass the plugin a parameter "quality" with the value "high", an attribute can be specified:

```
<embed src="catgame.swf" quality="high">
```

This would be equivalent to the following, when using an `<object>` element instead:

```
<object data="catgame.swf">
  <param name="quality" value="high">
</object>
```

§ 4.7.8. The object element

Categories:

[Flow content](#),
[Phrasing content](#),
[Embedded content](#),
[listed](#), [submittable](#), and [reassociateable form-associated element](#),
[Palpable content](#).

Contexts in which this element can be used:

Where [embedded content](#) is expected.

Content model:

Zero or more `<param>` elements, then, [transparent](#).

Tag omission in text/html:

Neither tag is omissible.

Content attributes:

[Global attributes](#)
data - Address of the resource
type - Type of embedded resource
typemustmatch - Whether the **type** attribute and the [Content-Type](#) value need to match for the resource to be used
name - Name of [nested browsing context](#)
form - Associates the control with a `<form>` element
width - Horizontal dimension
height - Vertical dimension

Allowed ARIA role attribute values:

[‘application’](#), [‘document’](#) or [‘img’](#) or [‘presentation’](#).

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)
 Any **aria-*** attributes [applicable to the allowed roles](#).

DOM interface:

```

interface HTMLObjectElement : HTMLElement {
    attribute DOMString data;
    attribute DOMString type;
    attribute boolean typeMustMatch;
    attribute DOMString name;
    readonly attribute HTMLFormElement? form;
    attribute DOMString width;
    attribute DOMString height;
    readonly attribute Document? contentDocument;
    readonly attribute WindowProxy? contentWindow;

    readonly attribute boolean willValidate;
    readonly attribute ValidityState validity;
    readonly attribute DOMString validationMessage;
    boolean checkValidity();
    boolean reportValidity();
    void setCustomValidity(DOMString error);

    legacycaller any (any... arguments);
};

```

Depending on the type of content instantiated by the `<object>` element, the node also supports other interfaces.

The `<object>` element can represent an external resource, which, depending on the type of the resource, will either be treated as an image, as a [nested browsing context](#), or as an external resource to be processed by a [plugin](#).

The **data** attribute, if present, specifies the address of the resource. If present, the attribute must be a [valid non-empty URL potentially surrounded by spaces](#).

⚠Warning! Authors who reference resources from other [origins](#) that they do not trust are urged to use the `typemustmatch` attribute defined below. Without that attribute, it is possible in certain cases for an attacker on the remote host to use the plugin mechanism to run arbitrary scripts, even if the author has used features such as the Flash "allowScriptAccess" parameter.

The **type** attribute, if present, specifies the type of the resource. If present, the attribute must be a [valid mime type](#).

At least one of either the `data` attribute or the `type` attribute must be present.

The **typeMustMatch** attribute is a [boolean attribute](#) whose presence indicates that the resource specified by the `data` attribute is only to be used if the value of the `type` attribute and the [Content-Type](#) of the aforementioned resource match.

The `typemustmatch` attribute must not be specified unless both the `data` attribute and the `type` attribute are present.

The **name** attribute, if present, must be a [valid browsing context name](#). The given value is used to name the [nested browsing context](#), if applicable.

Whenever one of the following conditions occur:

- the element is created,
- the element is popped off the [stack of open elements](#) of an [HTML parser](#) or [XML parser](#),
- the element is not on the [stack of open elements](#) of an [HTML parser](#) or [XML parser](#), and it is either [inserted into a document](#) or [removed from a document](#),
- the element's [node document](#) changes whether it is [fully active](#),
- one of the element's ancestor `<object>` elements changes to or from showing its [fallback content](#),
- the element's `classid` attribute is set, changed, or removed,
- the element's `classid` attribute is not present, and its `data` attribute is set, changed, or removed,

- neither the element's `classid` attribute nor its `data` attribute are present, and its `type` attribute is set, changed, or removed,
- the element changes from [being rendered](#) to not being rendered, or vice versa,

...the user agent must [queue a task](#) to run the following steps to (re)determine what the `<object>` element represents. This [task](#) being [queued](#) or actively running must [delay the load event](#) of the element's [node document](#).

1. If the user has indicated a preference that this `<object>` element's [fallback content](#) be shown instead of the element's usual behavior, then jump to the step below labeled *fallback*.

NOTE:

For example, a user could ask for the element's [fallback content](#) to be shown because that content uses a format that the user finds more accessible.

2. If the element has an ancestor [media element](#), or has an ancestor `<object>` element that is *not* showing its [fallback content](#), or if the element is not [in a Document](#) with a [browsing context](#), or if the element's [node document](#) is not [fully active](#), or if the element is still in the [stack of open elements](#) of an [HTML parser](#) or [XML parser](#), or if the element is not [being rendered](#), then jump to the step below labeled *fallback*.
3. If the `classid` attribute is present, and has a value that isn't the empty string, then: if the user agent can find a [plugin](#) suitable according to the value of the `classid` attribute, and either plugins aren't being sandboxed or that [plugin](#) can be [secured](#), then that [plugin should be used](#), and the value of the `data` attribute, if any, should be passed to the [plugin](#). If no suitable [plugin](#) can be found, or if the [plugin](#) reports an error, jump to the step below labeled *fallback*.
4. If the `data` attribute is present and its value is not the empty string, then:
 1. If the `type` attribute is present and its value is not a type that the user agent supports, and is not a type that the user agent can find a [plugin](#) for, then the user agent may jump to the step below labeled *fallback* without fetching the content to examine its real type.
 2. [Parse](#) the [URL](#) specified by the `data` attribute, relative to the element.
 3. If that failed, [fire a simple event](#) named *error* at the element, then jump to the step below labeled *fallback*.
 4. Let *request* be a new [request](#) whose [URL](#) is the [resulting URL string](#), [client](#) is the element's [node document](#)'s [Window object's environment settings object](#), [destination](#) is "unknown", [omit-Origin-header flag](#) is set if the element doesn't have a [browsing context scope origin](#), [credentials mode](#) is "include", and whose [use-URL-credentials flag](#) is set.
 5. [Fetch](#) *request*.

Fetching the resource must [delay the load event](#) of the element's [node document](#) until the [task](#) that is [queued](#) by the [networking task source](#) once the resource has been fetched (defined next) has been run.

6. If the resource is not yet available (e.g., because the resource was not available in the cache, so that loading the resource required making a request over the network), then jump to the step below labeled *fallback*. The [task](#) that is [queued](#) by the [networking task source](#) once the resource is available must restart this algorithm from this step. Resources can load incrementally; user agents may opt to consider a resource "available" whenever enough data has been obtained to begin processing the resource.
7. If the load failed (e.g., there was an HTTP 404 error, there was a DNS error), [fire a simple event](#) named *error* at the element, then jump to the step below labeled *fallback*.
8. Determine the *resource type*, as follows:
 1. Let the *resource type* be unknown.
 2. If the `<object>` element has a `type` attribute and a `typemustmatch` attribute, and the resource has [associated Content-Type metadata](#), and the type specified in [the resource's Content-Type metadata](#) is an [ASCII case-insensitive](#) match for the value of the element's `type` attribute, then let *resource type* be that type and jump to the step below labeled *handler*.
 3. If the `<object>` element has a `typemustmatch` attribute, jump to the step below labeled *handler*.
 4. If the user agent is configured to strictly obey Content-Type headers for this resource, and the resource has

associated [Content-Type metadata](#), then let the *resource type* be the type specified in [the resource's Content-Type metadata](#), and jump to the step below labeled *handler*.

⚠Warning! This can introduce a vulnerability, wherein a site is trying to embed a resource that uses a particular plugin, but the remote site overrides that and instead furnishes the user agent with a resource that triggers a different plugin with different security characteristics.

5. If there is a *type* attribute present on the `<object>` element, and that attribute's value is not a type that the user agent supports, but it *is* a type that a [plugin](#) supports, then let the *resource type* be the type specified in that *type* attribute, and jump to the step below labeled *handler*.
6. Run the appropriate set of steps from the following list:

↪ If the resource has [associated Content-Type metadata](#)

1. Let *binary* be false.
2. If the type specified in [the resource's Content-Type metadata](#) is "text/plain", and the result of applying the [rules for distinguishing if a resource is text or binary](#) to the resource is that the resource is not text/plain, then set *binary* to true.
3. If the type specified in [the resource's Content-Type metadata](#) is "application/octet-stream", then set *binary* to true.
4. If *binary* is false, then let the *resource type* be the type specified in [the resource's Content-Type metadata](#), and jump to the step below labeled *handler*.
5. If there is a *type* attribute present on the `<object>` element, and its value is not application/octet-stream, then run the following steps:
 1. If the attribute's value is a type that a [plugin](#) supports, or the attribute's value is a type that starts with "image/" that is not also an [XML MIME type](#), then let the *resource type* be the type specified in that *type* attribute.
 2. Jump to the step below labeled *handler*.

↪ Otherwise, if the resource does not have [associated Content-Type metadata](#)

1. If there is a *type* attribute present on the `<object>` element, then let the *tentative type* be the type specified in that *type* attribute.
Otherwise, let *tentative type* be the [computed type of the resource](#).
2. If *tentative type* is not application/octet-stream, then let *resource type* be *tentative type* and jump to the step below labeled *handler*.
7. If applying the [URL parser](#) algorithm to the [URL](#) of the specified resource (after any redirects) results in a [URL record](#) whose [path](#) component matches a pattern that a [plugin](#) supports, then let *resource type* be the type that the plugin can handle.

EXAMPLE 405

For example, a plugin might say that it can handle resources with [path](#) components that end with the four character string ".swf".

NOTE:

It is possible for this step to finish, or for one of the substeps above to jump straight to the next step, with *resource type* still being unknown. In both cases, the next step will trigger fallback.

9. *Handler*: Handle the content as given by the first of the following cases that matches:

↪ If the *resource type* is not a type that the user agent supports, but it *is* a type that a [plugin](#) supports

If plugins are being sandboxed and the plugin that supports *resource type* cannot be [secured](#), jump to the step below labeled *fallback*.

Otherwise, the user agent should use the plugin that supports *resource type* and pass the content of the resource to that [plugin](#). If the [plugin](#) reports an error, then jump to the step below labeled *fallback*.

↪ If the *resource type* is an **XML MIME type**, or if the *resource type* does not start with "image/"

The `<object>` element must be associated with a newly created [nested browsing context](#), if it does not already have one.

If the [URL](#) of the given resource is not about:blank, the element's [nested browsing context](#) must then be [navigated](#) to that resource, with [replacement enabled](#), and with the `<object>` element's [node document's browsing context](#) as the [source browsing context](#). (The data attribute of the `<object>` element doesn't get updated if the browsing context gets further navigated to other locations.)

If the [URL](#) of the given resource *is* about:blank, then, instead, the user agent must [queue a task to fire a simple event](#) named load at the `<object>` element. ■ No load event is fired at the about:blank document itself. ■

The `<object>` element [represents](#) the [nested browsing context](#).

If the name attribute is present, the [browsing context name](#) must be set to the value of this attribute; otherwise, the [browsing context name](#) must be set to the empty string.

↪ If the *resource type* starts with "image/", and support for images has not been disabled

Apply the [image sniffing](#) rules to determine the type of the image.

The `<object>` element [represents](#) the specified image. The image is not a [nested browsing context](#).

If the image cannot be rendered, e.g., because it is malformed or in an unsupported format, jump to the step below labeled *fallback*.

↪ Otherwise

The given *resource type* is not supported. Jump to the step below labeled *fallback*.

NOTE:

If the previous step ended with the *resource type* being unknown, this is the case that is triggered.

10. The element's contents are not part of what the `<object>` element represents.

11. Abort these steps. Once the resource is completely loaded, [queue a task to fire a simple event](#) named load at the element.

5. If the data attribute is absent but the type attribute is present, and the user agent can find a [plugin](#) suitable according to the value of the type attribute, and either plugins aren't being sandboxed or the [plugin](#) can be [secured](#), then that [plugin should be used](#). If these conditions cannot be met, or if the [plugin](#) reports an error, jump to the step below labeled *fallback*. Otherwise abort these steps; once the plugin is completely loaded, [queue a task to fire a simple event](#) named load at the element.

6. *Fallback*: The `<object>` element [represents](#) the element's children, ignoring any leading `<param>` element children. This is the element's [fallback content](#). If the element has an instantiated [plugin](#), then unload it.

When the algorithm above instantiates a [plugin](#), the user agent should pass to the [plugin](#) used the names and values of all the attributes on the element, in the order they were added to the element, with the attributes added by the parser being ordered in source order, followed by a parameter named "PARAM" whose value is null, followed by all the names and values of [parameters](#) given by `<param>` elements that are children of the `<object>` element, in [tree order](#). If the [plugin](#) supports a scriptable interface, the `HTMLObjectElement` object representing the element should expose that interface. The `<object>` element [represents](#) the [plugin](#). The [plugin](#) is not a [nested browsing context](#).

Plugins are considered sandboxed for the purpose of an `<object>` element if the [sandboxed plugins browsing context flag](#) is set on the `<object>` element's [node document's active sandboxing flag set](#).

Due to the algorithm above, the contents of `<object>` elements act as [fallback content](#), used only when referenced resources can't be shown (e.g., because it returned a 404 error). This allows multiple `<object>` elements to be nested inside each other,

targeting multiple user agents with different capabilities, with the user agent picking the first one it supports.

When an `<object>` element represents a [nested browsing context](#): if the `<object>` element's [nested browsing context's active document](#) is not [ready for post-load tasks](#), and when anything is [delaying the load event](#) of the `<object>` element's [browsing context's active document](#), and when the `<object>` element's [browsing context](#) is in the [delaying load events mode](#), the object must [delay the load event](#) of its document.

The [task source](#) for the [tasks](#) mentioned in this section is the [DOM manipulation task source](#).

Whenever the `name` attribute is set, if the `<object>` element has a nested [browsing context](#), its `name` must be changed to the new value. If the attribute is removed, if the `<object>` element has a [browsing context](#), the [browsing context name](#) must be set to the empty string.

The `form` attribute is used to explicitly associate the `<object>` element with its [form owner](#).

Constraint validation: `<object>` elements are always [barred from constraint validation](#).

The `<object>` element supports [dimension attributes](#).

The IDL attributes `data`, `type` and `name` each must [reflect](#) the respective content attributes of the same name. The `typeMustMatch` IDL attribute must [reflect](#) the `typemustmatch` content attribute.

The `contentDocument` IDL attribute must return the [Document](#) object of the [active document](#) of the `<object>` element's [nested browsing context](#), if any and if its [origin](#) is the [same origin-domain](#) as the [origin](#) specified by the [incumbent settings object](#), or null otherwise.

The `contentWindow` IDL attribute must return the [WindowProxy](#) object of the `<object>` element's [nested browsing context](#), if it has one; otherwise, it must return null.

The `willValidate`, `validity`, and `validationMessage` attributes, and the `checkValidity()`, `reportValidity()`, and `setCustomValidity()` methods, are part of the [constraint validation API](#). The `form` IDL attribute is part of the element's forms API.

All `<object>` elements have a **legacy caller operation**. If the `<object>` element has an instantiated [plugin](#) that supports a scriptable interface that defines a legacy caller operation, then that must be the behavior of the object's legacy caller operation. Otherwise, the object's legacy caller operation must be to throw a [NotSupportedError](#) exception.

EXAMPLE 406

In the following example, a Java applet is embedded in a page using the `object` element. (Generally speaking, it is better to avoid using applets like these and instead use native JavaScript and HTML to provide the functionality, since that way the application will work on all Web browsers without requiring a third-party plugin. Many devices, especially embedded devices, do not support third-party technologies like Java.)

```
<figure>
  <object type="application/x-java-applet">
    <param name="code" value="MyJavaClass">
    <p>You do not have Java available, or it is disabled.</p>
  </object>
  <figcaption>My Java Clock</figcaption>
</figure>
```

EXAMPLE 407

In this example, an HTML page is embedded in another using the `object` element.

```
<figure>
  <object data="clock.html"></object>
  <figcaption>My HTML Clock</figcaption>
</figure>
```

EXAMPLE 408

The following example shows how a plugin can be used in HTML (in this case the Flash plugin, to show a video file). Fallback is provided for users who do not have Flash enabled, in this case using the `<video>` element to show the video for those using user agents that support `<video>`, and finally providing a link to the video for those who have neither Flash nor a video-capable browser.

```
<p>Look at my video:
  <object type="application/x-shockwave-flash">
    <param name=movie value="https://video.example.com/library/watch.swf">
    <param name=allowfullscreen value=true>
    <param name=flashvars value="https://video.example.com/vids/315981">
    <video controls src="https://video.example.com/vids/315981">
      <a href="https://video.example.com/vids/315981">View video</a>.
    </video>
  </object>
</p>
```

§ 4.7.9. The param element**Categories:**

None.

Contexts in which this element can be used:

As a child of an `<object>` element, before any [flow content](#).

Content model:

[Nothing](#).

Tag omission in text/html:

No [end tag](#)

Content attributes:

[Global attributes](#)

name - Name of parameter

value - Value of parameter

Allowed [ARIA role attribute](#) values:

None

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

DOM interface:

```
interface HTMLParamElement : HTMLElement {
  attribute DOMString name;
  attribute DOMString value;
};
```

The `<param>` element defines parameters for plugins invoked by object elements. It does not [represent](#) anything on its own.

The **name** attribute gives the name of the parameter.

The **value** attribute gives the value of the parameter.

Both attributes must be present. They may have any value.

If both attributes are present, and if the parent element of the `param` is an `<object>` element, then the element defines a **parameter** with the given name-value pair.

If either the name or value of a [parameter](#) defined by a `<param>` element that is the child of an `<object>` element that [represents](#) an instantiated [plugin](#) changes, and if that [plugin](#) is communicating with the user agent using an API that features the

ability to update the [plugin](#) when the name or value of a [parameter](#) so changes, then the user agent must appropriately exercise that ability to notify the [plugin](#) of the change.

The IDL attributes **name** and **value** must both [reflect](#) the respective content attributes of the same name.

EXAMPLE 409

The following example shows how the `<param>` element can be used to pass a parameter to a plugin, in this case the O3D plugin.

```
<!DOCTYPE HTML>
<html lang="en">
  <head>
    <title>O3D Utah Teapot</title>
  </head>
  <body>
    <p>
      <object type="application/vnd.o3d.auto">
        <param name="o3d_features" value="FloatingPointTextures">
          

          <p>To see the teapot actually rendered by O3D on your
            computer, please download and install the <a
              href="https://code.google.com/apis/o3d/docs/gettingstarted.html#install">O3D
            plugin</a>.</p>
        </object>
        <script src="o3d-teapot.js"></script>
      </p>
    </body>
  </html>
```

§ 4.7.10. The video element

Categories:

[Flow content.](#)

[Phrasing content.](#)

[Embedded content.](#)

If the element has a [controls](#) attribute: [interactive content.](#)

[Palpable content.](#)

Contexts in which this element can be used:

Where [embedded content](#) is expected.

Content model:

If the element has a `src` attribute: zero or more `<track>` elements, then [transparent](#), but with no [media element](#) descendants.

If the element does not have a `src` attribute: zero or more `<source>` elements, then zero or more `<track>` elements, then [transparent](#), but with no [media element](#) descendants.

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

`src` - Address of the resource

`crossorigin` - How the element handles crossorigin requests

`poster` - Poster frame to show prior to video playback

[preload](#) - Hints how much buffering the [media resource](#) will likely need
[autoplay](#) - Hint that the [media resource](#) can be started automatically when the page is loaded
[loop](#) - Whether to loop the [media resource](#)
[muted](#) - Whether to mute the [media resource](#) by default
[controls](#) - Show user agent controls
[width](#) - Horizontal dimension
[height](#) - Vertical dimension

Allowed [ARIA role attribute](#) values:

[‘application’](#).

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any [aria-* attributes](#) [applicable to the allowed roles](#).

DOM interface:

```

interface HTMLVideoElement : HTMLMediaElement {
    attribute unsigned long width;
    attribute unsigned long height;
    readonly attribute unsigned long videoWidth;
    readonly attribute unsigned long videoHeight;
    attribute DOMString poster;
};
  
```

A [<video>](#) element is used for playing videos or movies, and audio files with captions.

Content may be provided inside the [<video>](#) element. User agents should not show this content to the user; it is intended for older Web browsers which do not support [<video>](#), so that legacy video plugins can be tried, or to show text to the users of these older browsers informing them of how to access the video contents.

NOTE:

In particular, this content is not intended to address accessibility concerns. To make video content accessible to the partially sighted, the blind, the hard-of-hearing, the deaf, and those with other physical or cognitive disabilities, a variety of features are available. Captions can be provided, either embedded in the video stream or as external files using the [<track>](#) element. Sign-language tracks can be provided, again either embedded in the video stream. Audio descriptions can be provided, either as a separate track embedded in the video stream, or in text form using a [WebVTT file](#) referenced using the [<track>](#) element and synthesized into speech by the user agent. WebVTT can also be used to provide chapter titles. For users who would rather not use a media element at all, transcripts or other textual alternatives can be provided by simply linking to them in the prose near the [<video>](#) element. [\[WEBVTT\]](#)

The [<video>](#) element is a [media element](#) whose [media data](#) is ostensibly video data, possibly with associated audio data.

The [src](#), [preload](#), [autoplay](#), [loop](#), [muted](#), and [controls](#) attributes are the attributes common to all media elements.

The **poster** content attribute gives the address of an image file that the user agent can show while no video data is available. The attribute, if present, must contain a [valid non-empty URL potentially surrounded by spaces](#).

If the specified resource is to be used, then, when the element is created or when the **poster** attribute is set, changed, or removed, the user agent must run the following steps to determine the element's **poster frame** (regardless of the value of the element's [show poster flag](#)):

1. If there is an existing instance of this algorithm running for this video element, abort that instance of this algorithm without changing the [poster frame](#).
2. If the **poster** attribute's value is the empty string or if the attribute is absent, then there is no [poster frame](#); abort these steps.
3. [Parse](#) the **poster** attribute's value relative to the element. If this fails, then there is no [poster frame](#); abort these steps.
4. Let [request](#) be a new [request](#) whose [URL](#) is the [resulting URL string](#), [client](#) is the element's [node document](#)'s [Window](#)

object's [environment settings object](#), [type](#) is "image", [destination](#) is "subresource", [credentials mode](#) is "include", and whose [use-URL-credentials flag](#) is set.

5. [Fetch request](#). This must [delay the load event](#) of the element's [node document](#).

6. If an image is thus obtained, the [poster frame](#) is that image. Otherwise, there is no [poster frame](#).

NOTE:

The image given by the poster attribute, the poster frame, is intended to be a representative frame of the video (typically one of the first non-blank frames) that gives the user an idea of what the video is like.



A [<video>](#) element represents what is given for the first matching condition in the list below:

- ↪ When no video data is available (the element's [readyState](#) attribute is either `HAVE_NOTHING`, or `HAVE_METADATA` but no video data has yet been obtained at all, or the element's [readyState](#) attribute is any subsequent value but the [media resource](#) does not have a video channel)

The [<video>](#) element [represents](#) its [poster frame](#), if any, or else transparent black with no [intrinsic dimensions](#).

- ↪ When the [<video>](#) element is [paused](#), the [current playback position](#) is the first frame of video, and the element's [show poster flag](#) is set

The [<video>](#) element [represents](#) its [poster frame](#), if any, or else the first frame of the video.

- ↪ When the [<video>](#) element is [paused](#), and the frame of video corresponding to the [current playback position](#) is not available (e.g., because the video is seeking or buffering)

- ↪ When the [<video>](#) element is neither [potentially playing](#) nor [paused](#) (e.g., when seeking or stalled)

The [<video>](#) element [represents](#) the last frame of the video to have been rendered.

- ↪ When the [<video>](#) element is [paused](#)

The [<video>](#) element [represents](#) the frame of video corresponding to the [current playback position](#).

- ↪ Otherwise (the [<video>](#) element has a video channel and is [potentially playing](#))

The [<video>](#) element [represents](#) the frame of video at the continuously increasing "[current](#)" [position](#). When the [current playback position](#) changes such that the last frame rendered is no longer the frame corresponding to the [current playback position](#) in the video, the new frame must be rendered.

Frames of video must be obtained from the video track that was [selected](#) when the [event loop](#) last reached step 1.

NOTE:

Which frame in a video stream corresponds to a particular playback position is defined by the video stream's format.

The [<video>](#) element also [represents](#) any [text track cues](#) whose [text track cue active flag](#) is set and whose [text track](#) is in the [showing](#) mode, and any audio from the [media resource](#), at the [current playback position](#).

Any audio associated with the [media resource](#) must, if played, be played synchronized with the [current playback position](#), at the element's [effective media volume](#). The user agent must play the audio from audio tracks that were enabled when the [event loop](#) last reached step 1.

In addition to the above, the user agent may provide messages to the user (such as "buffering", "no video loaded", "error", or more detailed information) by overlaying text or icons on the video or other areas of the element's playback area, or in another appropriate manner.

User agents that cannot render the video may instead make the element [represent](#) a link to an external video playback utility or to the video data itself.

When a [<video>](#) element's [media resource](#) has a video channel, the element [provides a paint source](#) whose width is the [media resource](#)'s [intrinsic width](#), whose height is the [media resource](#)'s [intrinsic height](#), and whose appearance is the frame of video corresponding to the [current playback position](#), if that is available, or else (e.g., when the video is seeking or buffering) its previous appearance, if any, or else (e.g., because the video is still loading the first frame) blackness.



This definition is non-normative. Implementation requirements are given below this definition.

`video`.`videoWidth`

`video`.`videoHeight`

These attributes return the intrinsic dimensions of the video, or zero if the dimensions are not known.

The **intrinsic width** and **intrinsic height** of the [media resource](#) are the dimensions of the resource in CSS pixels after taking into account the resource's dimensions, aspect ratio, clean aperture, resolution, and so forth, as defined for the format used by the resource. If an anamorphic format does not define how to apply the aspect ratio to the video data's dimensions to obtain the "correct" dimensions, then the user agent must apply the ratio by increasing one dimension and leaving the other unchanged.

The `videoWidth` IDL attribute must return the [intrinsic width](#) of the video in CSS pixels. The `videoHeight` IDL attribute must return the [intrinsic height](#) of the video in CSS pixels. If the element's `readyState` attribute is `HAVE_NOTHING`, then the attributes must return 0.

Whenever the [intrinsic width](#) or [intrinsic height](#) of the video changes (including, for example, because the selected video track was changed), if the element's `readyState` attribute is not `HAVE_NOTHING`, the user agent must [queue a task to fire a simple event](#) named `resize` at the [media element](#).

The `<video>` element supports [dimension attributes](#).

In the absence of style rules to the contrary, video content should be rendered inside the element's playback area such that the video content is shown centered in the playback area at the largest possible size that fits completely within it, with the video content's aspect ratio being preserved. Thus, if the aspect ratio of the playback area does not match the aspect ratio of the video, the video will be shown letterboxed or pillarboxed. Areas of the element's playback area that do not contain the video represent nothing.

NOTE:

In user agents that implement CSS, the above requirement can be implemented by using the style rule suggested in [§10 Rendering](#).

The [intrinsic width](#) of a `<video>` element's playback area is the [intrinsic width](#) of the [poster frame](#), if that is available and the element currently [represents](#) its poster frame; otherwise, it is the [intrinsic width](#) of the video resource, if that is available; otherwise the [intrinsic width](#) is missing.

The [intrinsic height](#) of a `<video>` element's playback area is the [intrinsic height](#) of the [poster frame](#), if that is available and the element currently [represents](#) its poster frame; otherwise it is the [intrinsic height](#) of the video resource, if that is available; otherwise the [intrinsic height](#) is missing.

The [default object size](#) is a width of 300 CSS pixels and a height of 150 CSS pixels. [\[CSS3-IMAGES\]](#)



User agents should provide controls to enable or disable the display of closed captions, audio description tracks, and other additional data associated with the video stream, though such features should, again, not interfere with the page's normal rendering.

User agents may allow users to view the video content in manners more suitable to the user (e.g., fullscreen or in an independent resizable window). As for the other user interface features, controls to enable this should not interfere with the page's normal rendering unless the user agent is [exposing a user interface](#). In such an independent context, however, user agents may make full user interfaces visible even if the [controls](#) attribute is absent.

User agents may allow video playback to affect system features that could interfere with the user's experience; for example, user agents could disable screensavers while video playback is in progress.



The **poster** IDL attribute must [reflect](#) the poster content attribute.

EXAMPLE 410

This example shows how to detect when a video has failed to play correctly:

```
<script>
function failed(e) {
  // video playback failed - show a message saying why
  switch (e.target.error.code) {
    case e.target.error.MEDIA_ERR_ABORTED:
      alert('You aborted the video playback.');
```

break;

```
    case e.target.error.MEDIA_ERR_NETWORK:
      alert('A network error caused the video download to fail part-way.');
```

break;

```
    case e.target.error.MEDIA_ERR_DECODE:
      alert('The video playback was aborted due to a corruption problem or because the video
used features your browser did not support.');
```

break;

```
    case e.target.error.MEDIA_ERR_SRC_NOT_SUPPORTED:
      alert('The video could not be loaded, either because the server or network failed or
because the format is not supported.');
```

break;

```
    default:
      alert('An unknown error occurred.');
```

break;

```
  }
}
</script>
<p><video src="tgif.vid" autoplay controls onerror="failed(event)"></video></p>
<p><a href="tgif.vid">Download the video file</a></p>
```

§ 4.7.11. The audio element

Categories:

[Flow content.](#)

[Phrasing content.](#)

[Embedded content.](#)

If the element has a [controls](#) attribute: [Interactive content.](#)

If the element has a [controls](#) attribute: [Palpable content.](#)

Contexts in which this element can be used:

Where [embedded content](#) is expected.

Content model:

If the element has a `src` attribute: zero or more [<track>](#) elements, then [transparent](#), but with no [media element](#) descendants.

If the element does not have a `src` attribute: zero or more [<source>](#) elements, then zero or more [<track>](#) elements, then [transparent](#), but with no [media element](#) descendants.

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

`src` - Address of the resource

`crossorigin` - How the element handles crossorigin requests

preload - Hints how much buffering the [media resource](#) will likely need

autoplay - Hint that the [media resource](#) can be started automatically when the page is loaded

loop - Whether to loop the [media resource](#)

muted - Whether to mute the [media resource](#) by default

[controls](#) - Show user agent controls

Allowed [ARIA role attribute](#) values:

[‘application’](#).

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
[NamedConstructor=Audio(optional DOMString src)]
interface HTMLAudioElement : HTMLMediaElement {};
```

An `<audio>` element [represents](#) a sound or audio stream.

Content may be provided inside the `<audio>` element. User agents should not show this content to the user; it is intended for older Web browsers which do not support `<audio>`, so that legacy audio plugins can be tried, or to show text to the users of these older browsers informing them of how to access the audio contents.

NOTE:

In particular, this content is not intended to address accessibility concerns. To make audio content accessible to the deaf or to those with other physical or cognitive disabilities, a variety of features are available. If captions or a sign language video are available, the `<video>` element can be used instead of the `<audio>` element to play the audio, allowing users to enable the visual alternatives. Chapter titles can be provided to aid navigation, using the `<track>` element and a [WebVTT file](#). And, naturally, transcripts or other textual alternatives can be provided by simply linking to them in the prose near the `<audio>` element. [\[WEBVTT\]](#)

The `<audio>` element is a [media element](#) whose [media data](#) is ostensibly audio data.

The `src`, `preload`, `autoplay`, `loop`, `muted`, and [controls](#) attributes are the attributes common to all media elements.

When an `<audio>` element is [potentially playing](#), it must have its audio data played synchronized with the [current playback position](#), at the element's [effective media volume](#). The user agent must play the audio from audio tracks that were enabled when the [event loop](#) last reached step 1.

When an `<audio>` element is not [potentially playing](#), audio must not play for the element.

This definition is non-normative. Implementation requirements are given below this definition.

```
audio = new Audio( [ url ] )
```

Returns a new `<audio>` element, with the `src` attribute set to the value passed in the argument, if applicable.

A constructor is provided for creating `HTMLAudioElement` objects (in addition to the factory methods from DOM such as `createElement()`): **Audio(src)**. When invoked as a constructor, it must return a new `HTMLAudioElement` object (a new audio element). The element must be created with its `preload` attribute set to the literal value "auto". If the `src` argument is present, the object created must be created with its `src` content attribute set to the provided value (this will cause the user agent to invoke the object's [resource selection algorithm](#) before returning). The element's [node document](#) must be the [active document](#) of the [browsing context](#) of the `Window` object on which the interface object of the invoked constructor is found.

§ 4.7.12. The source element

Categories:

None.

Contexts in which this element can be used:

As a child of a [media element](#), before any [flow content](#) or `<track>` elements.

Content model:

[Nothing](#).

Tag omission in text/html:

No [end tag](#)

Content attributes:

[Global attributes](#)

`src` - Address of the resource

`type` - Type of embedded resource

Allowed [ARIA role attribute](#) values:

None

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

DOM interface:

```
interface HTMLSourceElement : HTMLElement {  
    attribute DOMString src;  
    attribute DOMString type;  
};
```

The `<source>` element allows authors to specify multiple alternative [media resources](#) for [media elements](#). It does not [represent](#) anything on its own.

The `src` attribute gives the address of the [media resource](#). The value must be a [valid non-empty URL potentially surrounded by spaces](#). This attribute must be present.

NOTE:

Dynamically modifying a `<source>` element and its attribute when the element is already inserted in a video or `<audio>` element will have no effect. To change what is playing, just use the `src` attribute on the [media element](#) directly, possibly making use of the `canPlayType()` method to pick from amongst available resources. Generally, manipulating `<source>` elements manually after the document has been parsed is an unnecessarily complicated approach.

The `type` content attribute gives the type of the [media resource](#), to help the user agent determine if it can play this [media resource](#) before fetching it. If specified, its value must be a [valid MIME type](#). The codecs parameter, which certain MIME types define, might be necessary to specify exactly how the resource is encoded. [\[RFC6381\]](#)

EXAMPLE 411

The following list shows some examples of how to use the `codecs=` MIME parameter in the `type` attribute.

H.264 Constrained baseline profile video (main and extended video compatible) level 3 and Low-Complexity AAC audio in MP4 container

```
<source src='video.mp4' type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
```

H.264 Extended profile video (baseline-compatible) level 3 and Low-Complexity AAC audio in MP4 container

```
<source src='video.mp4' type='video/mp4; codecs="avc1.58A01E, mp4a.40.2"'>
```

H.264 Main profile video level 3 and Low-Complexity AAC audio in MP4 container

```
<source src='video.mp4' type='video/mp4; codecs="avc1.4D401E, mp4a.40.2"'>
```

H.264 "High" profile video (incompatible with main, baseline, or extended profiles) level 3 and Low-Complexity AAC audio in MP4 container

```
<source src='video.mp4' type='video/mp4; codecs="avc1.64001E, mp4a.40.2"'>
```

MPEG-4 Visual Simple Profile Level 0 video and Low-Complexity AAC audio in MP4 container

```
<source src='video.mp4' type='video/mp4; codecs="mp4v.20.8, mp4a.40.2"'>
```

MPEG-4 Advanced Simple Profile Level 0 video and Low-Complexity AAC audio in MP4 container

```
<source src='video.mp4' type='video/mp4; codecs="mp4v.20.240, mp4a.40.2"'>
```

MPEG-4 Visual Simple Profile Level 0 video and AMR audio in 3GPP container

```
<source src='video.3gp' type='video/3gpp; codecs="mp4v.20.8, samr"'>
```

Theora video and Vorbis audio in Ogg container

```
<source src='video.ogv' type='video/ogg; codecs="theora, vorbis"'>
```

Theora video and Speex audio in Ogg container

```
<source src='video.ogv' type='video/ogg; codecs="theora, speex"'>
```

Vorbis audio alone in Ogg container

```
<source src='audio.ogg' type='audio/ogg; codecs=vorbis'>
```

Speex audio alone in Ogg container

```
<source src='audio.spx' type='audio/ogg; codecs=speex'>
```

FLAC audio alone in Ogg container

```
<source src='audio.oga' type='audio/ogg; codecs=flac'>
```

Dirac video and Vorbis audio in Ogg container

```
<source src='video.ogv' type='video/ogg; codecs="dirac, vorbis"'>
```

If a source element is inserted as a child of a [media element](#) that has no `src` attribute and whose `networkState` has the value `NETWORK_EMPTY`, the user agent must invoke the [media element's resource selection algorithm](#).

The IDL attributes `src` and `type` must [reflect](#) the respective content attributes of the same name.

EXAMPLE 412

If the author isn't sure if user agents will all be able to render the media resources provided, the author can listen to the error event on the last [source](#) element and trigger fallback behavior:

```
<script>
function fallback(video) {
  // replace <video> with its contents
  while (video.hasChildNodes()) {
    if (video.firstChild instanceof HTMLSourceElement)
      video.removeChild(video.firstChild);
    else
      video.parentNode.insertBefore(video.firstChild, video);
  }
  video.parentNode.removeChild(video);
}
</script>
<video controls autoplay>
  <source src='video.mp4' type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
  <source src='video.ogv' type='video/ogg; codecs="theora, vorbis"'
    onerror="fallback(parentNode)">
  ...
</video>
```

4.7.13. The track element

Categories:

None.

Contexts in which this element can be used:

As a child of a [media element](#), before any [flow content](#).

Content model:

Nothing.

Tag omission in text/html:

No [end tag](#)

Content attributes:

[Global attributes](#)

`kind` - The type of text track

`src` - Address of the resource

`srclang` - Language of the text track

`label` - User-visible label

`default` - Enable the track if no other [text track](#) is more suitable

Allowed [ARIA role attribute](#) values:

None

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

DOM interface:

```
interface HTMLTrackElement : HTMLElement {
    attribute DOMString kind;
    attribute DOMString src;
    attribute DOMString srclang;
    attribute DOMString label;
    attribute boolean default;

    const unsigned short NONE = 0;
    const unsigned short LOADING = 1;
    const unsigned short LOADED = 2;
    const unsigned short ERROR = 3;
    readonly attribute unsigned short readyState;

    readonly attribute TextTrack track;
};
```

The `<track>` element allows authors to specify explicit external text resources for [media elements](#). It does not [represent](#) anything on its own.

The `kind` attribute is an [enumerated attribute](#). The following table lists the keywords defined for this attribute. The keyword given in the first cell of each row maps to the state given in the second cell.

Keyword	State	Brief description
subtitles	Subtitles	Transcription or translation of the dialog, suitable for when the sound is available but not understood (e.g., because the user does not understand the language of the media resource 's audio track). Overlaid on the video.
captions	Captions	Transcription or translation of the dialog, sound effects, relevant musical cues, and other relevant audio information, suitable for when sound is unavailable or not clearly audible (e.g., because it is muted, drowned-out by ambient noise, or because the user is deaf). Overlaid on the video; labeled as appropriate for the hard-of-hearing.
descriptions	Descriptions	Textual descriptions of the video component of the media resource , intended for audio synthesis when the visual component is obscured, unavailable, or not usable (e.g., because the user is interacting with the application without a screen while driving, or because the user is blind). Synthesized as audio.
chapters	Chapters	Chapter titles, intended to be used for navigating the media resource . Displayed as an interactive (potentially nested) list in the user agent's interface.
metadata	Metadata	Tracks intended for use from script. Not displayed by the user agent.

The attribute may be omitted. The *missing value default* is the [subtitles](#) state. The *invalid value default* is the [metadata](#) state.

The `src` attribute gives the address of the text track data. The value must be a [valid non-empty URL potentially surrounded by spaces](#). This attribute must be present.

If the element has a `src` attribute whose value is not the empty string and whose value, when the attribute was set, could be successfully [parsed](#) relative to the element's [node document](#), then the element's **track URL** is the [resulting URL string](#). Otherwise, the element's **track URL** is the empty string.

If the element's **track URL** identifies a [WebVTT](#) resource, and the element's `kind` attribute is not in the [Metadata](#) state, then the [WebVTT](#) file must be a [WebVTT file using cue text](#). [\[WEBVTT\]](#)

Furthermore, if the element's **track URL** identifies a [WebVTT](#) resource, and the element's `kind` attribute is in the [chapters](#) state, then the [WebVTT](#) file must be both a [WebVTT file using chapter title text](#) and a [WebVTT file using only nested cues](#). [\[WEBVTT\]](#)

The **srcLang** attribute gives the language of the text track data. The value must be a valid BCP 47 language tag. This attribute must be present if the element's **kind** attribute is in the [subtitles](#) state. [\[BCP47\]](#)

If the element has a **srcLang** attribute whose value is not the empty string, then the element's **track language** is the value of the attribute. Otherwise, the element has no [track language](#).

The **label** attribute gives a user-readable title for the track. This title is used by user agents when listing [subtitle](#), [caption](#), and [audio description](#) tracks in their user interface.

The value of the **label** attribute, if the attribute is present, must not be the empty string. Furthermore, there must not be two track element children of the same [media element](#) whose **kind** attributes are in the same state, whose **srcLang** attributes are both missing or have values that represent the same language, and whose **label** attributes are again both missing or both have the same value.

If the element has a **label** attribute whose value is not the empty string, then the element's **track label** is the value of the attribute. Otherwise, the element's [track label](#) is an empty string.

The **default** attribute is a [boolean attribute](#), which, if specified, indicates that the track is to be enabled if the user's preferences do not indicate that another track would be more appropriate.

Each [media element](#) must have no more than one [<track>](#) element child whose **kind** attribute is in the [Subtitles](#) or [Captions](#) state and whose [default](#) attribute is specified.

Each [media element](#) must have no more than one [<track>](#) element child whose **kind** attribute is in the [Descriptions](#) state and whose [default](#) attribute is specified.

Each [media element](#) must have no more than one [<track>](#) element child whose **kind** attribute is in the [Chapters](#) state and whose [default](#) attribute is specified.

NOTE:

There is no limit on the number of [<track>](#) elements whose **kind** attribute is in the [Metadata](#) state and whose [default](#) attribute is specified.

This definition is non-normative. Implementation requirements are given below this definition.

track . readyState

Returns the [text track readiness state](#), represented by a number from the following list:

track . NONE (0)

The [text track not loaded](#) state.

track . LOADING (1)

The [text track loading](#) state.

track . LOADED (2)

The [text track loaded](#) state.

track . ERROR (3)

The [text track failed to load](#) state.

track . track

Returns the [TextTrack](#) object corresponding to the [text track](#) of the [<track>](#) element.

The **readyState** attribute must return the numeric value corresponding to the [text track readiness state](#) of the [<track>](#) element's [text track](#), as defined by the following list:

NONE (numeric value 0)

The [text track not loaded](#) state.

LOADING (numeric value 1)

The [text track loading](#) state.

LOADED (numeric value 2)

The [text track loaded](#) state.

ERROR (numeric value 3)

The [text track failed to load](#) state.

The **track** IDL attribute must, on getting, return the [<track>](#) element's [text track](#)'s corresponding [TextTrack](#) object.

The **src**, **srclang**, **label**, and **default** IDL attributes must [reflect](#) the respective content attributes of the same name. The **kind** IDL attribute must [reflect](#) the content attribute of the same name, [limited to only known values](#).

EXAMPLE 413

This video has subtitles in several languages:

```
<video src="brave.webm">
  <track kind=subtitles src=brave.en.vtt srclang=en label="English">
  <track kind=captions src=brave.en.hoh.vtt srclang=en label="English for the Hard of
Hearing">
  <track kind=subtitles src=brave.fr.vtt srclang=fr lang=fr label="Français">
  <track kind=subtitles src=brave.de.vtt srclang=de lang=de label="Deutsch">
</video>
```

(The [lang](#) attributes on the last two describe the language of the **label** attribute, not the language of the subtitles themselves. The language of the subtitles is given by the **srclang** attribute.)

§ 4.7.14. Media elements

[HTMLMediaElement](#) objects ([<audio>](#) and [<video>](#), in this specification) are simply known as **media elements**.

```
enum CanPlayTypeResult { "" /* empty string */, "maybe", "probably" };

typedef (MediaStream or MediaSource or Blob) MediaProvider;
```

```

interface HTMLMediaElement : HTMLElement {

    // error state
    readonly attribute MediaError? error;

    // network state
    attribute DOMString src;
    attribute MediaProvider? srcObject;
    readonly attribute DOMString currentSrc;
    attribute DOMString? crossOrigin;
    const unsigned short NETWORK_EMPTY = 0;
    const unsigned short NETWORK_IDLE = 1;
    const unsigned short NETWORK_LOADING = 2;
    const unsigned short NETWORK_NO_SOURCE = 3;
    readonly attribute unsigned short networkState;
    attribute DOMString preload;
    readonly attribute TimeRanges buffered;
    void load();
    CanPlayTypeResult canPlayType(DOMString type);

    // ready state
    const unsigned short HAVE_NOTHING = 0;
    const unsigned short HAVE_METADATA = 1;
    const unsigned short HAVE_CURRENT_DATA = 2;
    const unsigned short HAVE_FUTURE_DATA = 3;
    const unsigned short HAVE_ENOUGH_DATA = 4;
    readonly attribute unsigned short readyState;
    readonly attribute boolean seeking;

    // playback state
    attribute double currentTime;
    void fastSeek(double time);
    readonly attribute unrestricted double duration;
    object getStartDate();
    readonly attribute boolean paused;
    attribute double defaultPlaybackRate;
    attribute double playbackRate;
    readonly attribute TimeRanges played;
    readonly attribute TimeRanges seekable;
    readonly attribute boolean ended;
    attribute boolean autoplay;
    attribute boolean loop;
    void play();
    void pause();

    // controls
    attribute boolean controls;
    attribute double volume;
    attribute boolean muted;
    attribute boolean defaultMuted;

    // tracks
    [SameObject] readonly attribute AudioTrackList audioTracks;
    [SameObject] readonly attribute VideoTrackList videoTracks;
    [SameObject] readonly attribute TextTrackList textTracks;
    TextTrack addTextTrack(TextTrackKind kind, optional DOMString label = "", optional DOMString
language = "");
};

```

The **media element attributes**, `src`, `crossorigin`, `preload`, `autoplay`, `loop`, `muted`, and `controls`, apply to all [media elements](#). They are defined in this section.

[Media elements](#) are used to present audio data, or video and audio data, to the user. This is referred to as **media data** in this

section, since this section applies equally to [media elements](#) for audio or for video.

The term **media resource** is used to refer to the complete set of media data, e.g., the complete video file, or complete audio file.

A [media resource](#) can have multiple audio and video tracks. For the purposes of a [media element](#), the video data of the [media resource](#) is only that of the currently selected track (if any) as given by the element's `videoTracks` attribute when the [event loop](#) last reached step 1, and the audio data of the [media resource](#) is the result of mixing all the currently enabled tracks (if any) given by the element's `audioTracks` attribute when the [event loop](#) last reached step 1.

NOTE:

Both audio and `<video>` elements can be used for both audio and video. The main difference between the two is simply that the `<audio>` element has no playback area for visual content (such as video or captions), whereas the video element does.

Except where otherwise explicitly specified, the [task source](#) for all the tasks [queued](#) in this section and its subsections is the **media element event task source** of the [media element](#) in question.

4.7.14.1. Error codes

This definition is non-normative. Implementation requirements are given below this definition.

media . error

Returns a `MediaError` object representing the current error state of the element.

Returns null if there is no error.

All [media elements](#) have an associated error status, which records the last error the element encountered since its [resource selection algorithm](#) was last invoked. The **error** attribute, on getting, must return the `MediaError` object created for this last error, or null if there has not been an error.

```
interface MediaError {
  const unsigned short MEDIA_ERR_ABORTED = 1;
  const unsigned short MEDIA_ERR_NETWORK = 2;
  const unsigned short MEDIA_ERR_DECODE = 3;
  const unsigned short MEDIA_ERR_SRC_NOT_SUPPORTED = 4;
  readonly attribute unsigned short code;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

media . error . code

Returns the current error's error code, from the list below.

The **code** attribute of a `MediaError` object must return the code for the error, which must be one of the following:

MEDIA_ERR_ABORTED (numeric value 1)

The fetching process for the [media resource](#) was aborted by the user agent at the user's request.

MEDIA_ERR_NETWORK (numeric value 2)

A network error of some description caused the user agent to stop fetching the [media resource](#), after the resource was established to be usable.

MEDIA_ERR_DECODE (numeric value 3)

An error of some description occurred while decoding the [media resource](#), after the resource was established to be usable.

MEDIA_ERR_SRC_NOT_SUPPORTED (numeric value 4)

The [media resource](#) indicated by the `src` attribute or [assigned media provider object](#) was not suitable.

4.7.14.2. Location of the media resource

The **src** content attribute on [media elements](#) gives the address of the media resource (video, audio) to show. The attribute, if present, must contain a [valid non-empty URL potentially surrounded by spaces](#).

The **crossorigin** content attribute on [media elements](#) is a [CORS settings attribute](#).

If a [media element](#) is created with a **src** attribute, the user agent must [immediately](#) invoke the [media element's resource selection algorithm](#).

If a **src** attribute of a [media element](#) is set or changed, the user agent must invoke the [media element's media element load algorithm](#). (Removing the **src** attribute does not do this, even if there are [<source>](#) elements present.)

The **src** IDL attribute on [media elements](#) must [reflect](#) the content attribute of the same name.

The **crossOrigin** IDL attribute must [reflect](#) the **crossorigin** content attribute.

A **media provider object** is an object that can represent a [media resource](#), separate from a [URL](#). [MediaStream](#) objects, [MediaSource](#) objects, [Blob](#) objects, and [File](#) objects are all [media provider objects](#).

Each [media element](#) can have an **assigned media provider object**, which is a [media provider object](#). When a [media element](#) is created, it has no [assigned media provider object](#).

This definition is non-normative. Implementation requirements are given below this definition.

media . srcObject [= *source*]

Allows the [media element](#) to be assigned a [media provider object](#).

media . currentSrc

Returns the [URL](#) of the current [media resource](#), if any.

Returns the empty string when there is no [media resource](#), or it doesn't have a [URL](#).

The **currentSrc** IDL attribute is initially the empty string. Its value is changed by the [resource selection algorithm](#) defined below.

The **srcObject** IDL attribute, on getting, must return the element's [assigned media provider object](#), if any, or null otherwise. On setting, it must set the element's [assigned media provider object](#) to the new value, and then invoke the element's [media element load algorithm](#).

NOTE:

There are three ways to specify a [media resource](#), the **srcObject** IDL attribute, the **src** content attribute, and [<source>](#) elements. The IDL attribute takes priority, followed by the content attribute, followed by the elements.

4.7.14.3. MIME types

A [media resource](#) can be described in terms of its *type*, specifically a [MIME type](#), in some cases with a codecs parameter. (Whether the codecs parameter is allowed or not depends on the MIME type.) [\[RFC6381\]](#)

Types are usually somewhat incomplete descriptions; for example "video/mpeg" doesn't say anything except what the container type is, and even a type like "video/mp4; codecs="avc1.42E01E, mp4a.40.2"" doesn't include information like the actual bitrate (only the maximum bitrate). Thus, given a type, a user agent can often only know whether it *might* be able to play media of that type (with varying levels of confidence), or whether it definitely *cannot* play media of that type.

A **type that the user agent knows it cannot render** is one that describes a resource that the user agent definitely does not support, for example because it doesn't recognize the container type, or it doesn't support the listed codecs.

The [MIME type](#) "application/octet-stream" with no parameters is never [a type that the user agent knows it cannot render](#). User agents must treat that type as equivalent to the lack of any explicit [Content-Type metadata](#) when it is used to label a potential [media resource](#).

NOTE:

Only the [MIME type](#) "application/octet-stream" with no parameters is special-cased here; if any parameter appears with it, it will be treated just like any other [MIME type](#). This is a deviation from the rule that unknown [MIME type](#) parameters should be ignored.

This definition is non-normative. Implementation requirements are given below this definition.

`media . canPlayType(type)`

Returns the empty string (a negative response), "maybe", or "probably" based on how confident the user agent is that it can play media resources of the given type.

The `canPlayType(type)` method must return **the empty string** if *type* is [a type that the user agent knows it cannot render](#) or is the type "application/octet-stream"; it must return **"probably"** if the user agent is confident that the type represents a [media resource](#) that it can render if used in with this audio or [<video>](#) element; and it must return **"maybe"** otherwise. Implementors are encouraged to return "maybe" unless the type can be confidently established as being supported or not. Generally, a user agent should never return "probably" for a type that allows the codecs parameter if that parameter is not present.

EXAMPLE 414

This script tests to see if the user agent supports a (fictional) new format to dynamically decide whether to use a [<video>](#) element or a plugin:

```
<section id="video">
  <p><a href="playing-cats.nfv">Download video</a></p>
</section>
<script>
  var videoSection = document.getElementById('video');
  var videoElement = document.createElement('video');
  var support = videoElement.canPlayType('video/x-new-fictional-
format;codecs="kittens,bunnies"');
  if (support != "probably" && "New Fictional Video Plugin" in navigator.plugins) {
    // not confident of browser support
    // but we have a plugin
    // so use plugin instead
    videoElement = document.createElement("embed");
  } else if (support == "") {
    // no support from browser and no plugin
    // do nothing
    videoElement = null;
  }
  if (videoElement) {
    while (videoSection.hasChildNodes())
      videoSection.removeChild(videoSection.firstChild);
    videoElement.setAttribute("src", "playing-cats.nfv");
    videoSection.appendChild(videoElement);
  }
</script>
```

NOTE:

The `type` attribute of the [<source>](#) element allows the user agent to avoid downloading resources that use formats it cannot render.

 4.7.14.4. Network states

This definition is non-normative. Implementation requirements are given below this definition.

`media . networkState`

Returns the current state of network activity for the element, from the codes in the list below.

As [media elements](#) interact with the network, their current network activity is represented by the **networkState** attribute. On getting, it must return the current network state of the element, which must be one of the following values:

NETWORK_EMPTY (numeric value 0)

The element has not yet been initialized. All attributes are in their initial states.

NETWORK_IDLE (numeric value 1)

The element's [resource selection algorithm](#) is active and has selected a [resource](#), but it is not actually using the network at this time.

NETWORK_LOADING (numeric value 2)

The user agent is actively trying to download data.

NETWORK_NO_SOURCE (numeric value 3)

The element's [resource selection algorithm](#) is active, but it has not yet found a [resource](#) to use.

The [resource selection algorithm](#) defined below describes exactly when the networkState attribute changes value and what events fire to indicate changes in this state.

4.7.14.5. Loading the media resource

This definition is non-normative. Implementation requirements are given below this definition.

media . load()

Causes the element to reset and start selecting and loading a new [media resource](#) from scratch.

All [media elements](#) have an **autoplaying flag**, which must begin in the true state, and a **delaying-the-load-event flag**, which must begin in the false state. While the [delaying-the-load-event flag](#) is true, the element must [delay the load event](#) of its document.

When the **load()** method on a [media element](#) is invoked, the user agent must run the [media element load algorithm](#).

The **media element load algorithm** consists of the following steps.

1. Abort any already-running instance of the [resource selection algorithm](#) for this element.
2. If there are any [tasks](#) from the [media element](#)'s [media element event task source](#) in one of the [task queues](#), then remove those tasks.

NOTE:

Basically, pending events and callbacks for the media element are discarded when the media element starts loading a new resource.

3. If the [media element](#)'s networkState is set to NETWORK_LOADING or NETWORK_IDLE, [queue a task](#) to [fire a simple event](#) named abort at the [media element](#).
4. If the [media element](#)'s networkState is not set to NETWORK_EMPTY, then run these substeps:
 1. [Queue a task](#) to [fire a simple event](#) named emptied at the [media element](#).
 2. If a fetching process is in progress for the [media element](#), the user agent should stop it.
 3. If the [media element](#)'s [assigned media provider object](#) is a MediaSource object, then [detach it](#).
 4. [Forget the media element's media-resource-specific tracks](#).
 5. If readyState is not set to HAVE_NOTHING, then set it to that state.
 6. If the paused attribute is false, then set it to true.
 7. If seeking is true, set it to false.
 8. Set the [current playback position](#) to 0.

Set the [official playback position](#) to 0.

If this changed the [official playback position](#), then [queue a task to fire a simple event](#) named `timeupdate` at the [media element](#).

9. Set the [initial playback position](#) to 0.

10. Set the [timeline offset](#) to Not-a-Number (NaN).

11. Update the `duration` attribute to Not-a-Number (NaN).

NOTE:

The user agent will not fire a `durationchange` event for this particular change of the duration.

5. Set the `playbackRate` attribute to the value of the `defaultPlaybackRate` attribute.

6. Set the `error` attribute to null and the [autoplaying flag](#) to true.

7. Invoke the [media element's resource selection algorithm](#).

NOTE:

Playback of any previously playing [media resource](#) for this element stops.

The **resource selection algorithm** for a [media element](#) is as follows. This algorithm is always invoked as part of a [task](#), but one of the first steps in the algorithm is to return and continue running the remaining steps [in parallel](#). In addition, this algorithm interacts closely with the [event loop](#) mechanism; in particular, it has [synchronous sections](#) (which are triggered as part of the [event loop](#) algorithm). Steps in such sections are marked with .

1. Set the element's `networkState` attribute to the `NETWORK_NO_SOURCE` value.

2. Set the element's [show poster flag](#) to true.

3. Set the [media element's delaying-the-load-event flag](#) to true (this [delays the load event](#)).

4. [in parallel](#) await a stable state, allowing the [task](#) that invoked this algorithm to continue. The [synchronous section](#) consists of all the remaining steps of this algorithm until the algorithm says the [synchronous section](#) has ended. (Steps in [synchronous sections](#) are marked with .)

5.  If the [media element's blocked-on-parser flag](#) is false, then [populate the list of pending text tracks](#).

6.  If the [media element](#) has an [assigned media provider object](#), then let *mode* be *object*.

 Otherwise, if the [media element](#) has no [assigned media provider object](#) but has a `src` attribute, then let *mode* be *attribute*.

 Otherwise, if the [media element](#) does not have an [assigned media provider object](#) and does not have a `src` attribute, but does have a `<source>` element child, then let *mode* be *children* and let *candidate* be the first such `<source>` element child in [tree order](#).

 Otherwise the [media element](#) has no [assigned media provider object](#) and has neither a `src` attribute nor a `<source>` element child: set the `networkState` to `NETWORK_EMPTY`, and abort these steps; the [synchronous section](#) ends.

7.  Set the [media element's networkState](#) to `NETWORK_LOADING`.

8.  [Queue a task to fire a simple event](#) named `loadstart` at the [media element](#).

9. Run the appropriate steps from the following list:

↪ If *mode* is *object*

1.  Set the `currentSrc` attribute to the empty string.

2. End the [synchronous section](#), continuing the remaining steps [in parallel](#).

3. Run the [resource fetch algorithm](#) with the [assigned media provider object](#). If that algorithm returns with-

out aborting *this* one, then the load failed.

4. *Failed with media provider*: Reaching this step indicates that the media resource failed to load. [Queue a task](#) to run the [dedicated media source failure steps](#).
5. Wait for the [task](#) queued by the previous step to have executed.
6. Abort these steps. The element won't attempt to load another resource until this algorithm is triggered again.

↪ **If *mode* is *attribute***

1. ⌚ If the `src` attribute's value is the empty string, then end the [synchronous section](#), and jump down to the *failed with attribute* step below.
2. ⌚ Let *absolute URL* be the [absolute URL](#) that would have resulted from [parsing](#) the [URL](#) specified by the `src` attribute's value relative to the [media element](#) when the `src` attribute was last changed.
3. ⌚ If *absolute URL* was obtained successfully, set the `currentSrc` attribute to *absolute URL*.
4. End the [synchronous section](#), continuing the remaining steps [in parallel](#).
5. If *absolute URL* was obtained successfully, run the [resource fetch algorithm](#) with *absolute URL*. If that algorithm returns without aborting *this* one, then the load failed.
6. *Failed with attribute*: Reaching this step indicates that the media resource failed to load or that the given [URL](#) could not be [resolved](#). [Queue a task](#) to run the [dedicated media source failure steps](#).
7. Wait for the [task](#) queued by the previous step to have executed.
8. Abort these steps. The element won't attempt to load another resource until this algorithm is triggered again.

↪ **Otherwise (*mode* is *children*)**

1. ⌚ Let *pointer* be a position defined by two adjacent nodes in the [media element](#)'s child list, treating the start of the list (before the first child in the list, if any) and end of the list (after the last child in the list, if any) as nodes in their own right. One node is the node before *pointer*, and the other node is the node after *pointer*. Initially, let *pointer* be the position between the *candidate* node and the next node, if there are any, or the end of the list, if it is the last node.

As [nodes are inserted](#) and [removed](#) into the [media element](#), *pointer* must be updated as follows:

If a new node is inserted between the two nodes that define *pointer*

Let *pointer* be the point between the node before *pointer* and the new node. In other words, insertions at *pointer* go after *pointer*.

If the node before *pointer* is removed

Let *pointer* be the point between the node after *pointer* and the node before the node after *pointer*. In other words, *pointer* doesn't move relative to the remaining nodes.

If the node after *pointer* is removed

Let *pointer* be the point between the node before *pointer* and the node after the node before *pointer*. Just as with the previous case, *pointer* doesn't move relative to the remaining nodes.

Other changes don't affect *pointer*.

2. ⌚ *Process candidate*: If *candidate* does not have a `src` attribute, or if its `src` attribute's value is the empty string, then end the [synchronous section](#), and jump down to the *failed with elements* step below.
3. ⌚ Let *absolute URL* be the [absolute URL](#) that would have resulted from [parsing](#) the [URL](#) specified by *candidate*'s `src` attribute's value relative to the *candidate* when the `src` attribute was last changed.
4. ⌚ If *absolute URL* was not obtained successfully, then end the [synchronous section](#), and jump down to the *failed with elements* step below.
5. ⌚ If *candidate* has a `type` attribute whose value, when parsed as a [MIME type](#) (including any codecs

- described by the `codecs` parameter, for types that define that parameter), represents [a type that the user agent knows it cannot render](#), then end the [synchronous section](#), and jump down to the *failed with elements* step below.
6. ⌚ Set the `currentSrc` attribute to `absolute URL`.
 7. End the [synchronous section](#), continuing the remaining steps [in parallel](#).
 8. Run the [resource fetch algorithm](#) with `absolute URL`. If that algorithm returns without aborting *this* one, then the load failed.
 9. *Failed with elements*: [Queue a task](#) to [fire a simple event](#) named `error` at the `candidate` element.
 10. [Await a stable state](#). The [synchronous section](#) consists of all the remaining steps of this algorithm until the algorithm says the [synchronous section](#) has ended. (Steps in [synchronous sections](#) are marked with ⌚.)
 11. ⌚ [Forget the media element's media-resource-specific tracks](#).
 12. ⌚ *Find next candidate*: Let `candidate` be null.
 13. ⌚ *Search loop*: If the node after `pointer` is the end of the list, then jump to the *waiting* step below.
 14. ⌚ If the node after `pointer` is a `<source>` element, let `candidate` be that element.
 15. ⌚ Advance `pointer` so that the node before `pointer` is now the node that was after `pointer`, and the node after `pointer` is the node after the node that used to be after `pointer`, if any.
 16. ⌚ If `candidate` is null, jump back to the *search loop* step. Otherwise, jump back to the *process candidate* step.
 17. ⌚ *Waiting*: Set the element's `networkState` attribute to the `NETWORK_NO_SOURCE` value.
 18. ⌚ Set the element's [show poster flag](#) to true.
 19. ⌚ [Queue a task](#) to set the element's [delaying-the-load-event flag](#) to false. This stops [delaying the load event](#).
 20. End the [synchronous section](#), continuing the remaining steps [in parallel](#).
 21. Wait until the node after `pointer` is a node other than the end of the list. (This step might wait forever.)
 22. [Await a stable state](#). The [synchronous section](#) consists of all the remaining steps of this algorithm until the algorithm says the [synchronous section](#) has ended. (Steps in [synchronous sections](#) are marked with ⌚.)
 23. ⌚ Set the element's [delaying-the-load-event flag](#) back to true (this [delays the load event](#) again, in case it hasn't been fired yet).
 24. ⌚ Set the `networkState` back to `NETWORK_LOADING`.
 25. ⌚ Jump back to the *find next candidate* step above.

The **dedicated media source failure steps** are the following steps:

1. Set the `error` attribute to a new `MediaError` object whose `code` attribute is set to `MEDIA_ERR_SRC_NOT_SUPPORTED`.
2. [Forget the media element's media-resource-specific tracks](#).
3. Set the element's `networkState` attribute to the `NETWORK_NO_SOURCE` value.
4. Set the element's [show poster flag](#) to true.
5. [Fire a simple event](#) named `error` at the [media element](#).
6. Set the element's [delaying-the-load-event flag](#) to false. This stops [delaying the load event](#).

The **resource fetch algorithm** for a [media element](#) and a given [absolute URL](#) or [media provider object](#) is as follows:

1. If the algorithm was invoked with a [URL](#), then let *mode* be *remote*, otherwise let *mode* be *local*.
2. If *mode* is *remote*, then let the *current media resource* be the resource given by the [absolute URL](#) passed to this algorithm; otherwise, let the *current media resource* be the resource given by the [media provider object](#). Either way, the *current media resource* is now the element's [media resource](#).
3. Remove all [media-resource-specific text tracks](#) from the [media element](#)'s [list of pending text tracks](#), if any.
4. Run the appropriate steps from the following list:

↪ **If *mode* is remote**

1. Optionally, run the following substeps. This is the expected behavior if the user agent intends to not attempt to fetch the resource until the user requests it explicitly (e.g., as a way to implement the `preload` attribute's `none` keyword).
 1. Set the `networkState` to `NETWORK_IDLE`.
 2. [Queue a task](#) to [fire a simple event](#) named `suspend` at the element.
 3. [Queue a task](#) to set the element's [delaying-the-load-event flag](#) to `false`. This stops [delaying the load event](#).
 4. Wait for the task to be run.
 5. Wait for an implementation-defined event (e.g., the user requesting that the media element begin playback).
 6. Set the element's [delaying-the-load-event flag](#) back to `true` (this [delays the load event](#) again, in case it hasn't been fired yet).
 7. Set the `networkState` to `NETWORK_LOADING`.
2. Let *request* be the result of [creating a potential-CORS request](#) given *current media resource*'s [absolute URL](#) and the [media element](#)'s `crossorigin` content attribute value.

Set *request*'s `client` to the [media element](#)'s [node document](#)'s `Window` object's [environment settings object](#) and `type` to `"audio"` if the [media element](#) is an `audio` element and to `"video"` otherwise.

[Fetch](#) *request*.

The *response*'s unsafe response obtained in this fashion, if any, contains the [media data](#). It can be [CORS-same-origin](#) or [CORS-cross-origin](#); this affects whether subtitles referenced in the [media data](#) are exposed in the API and, for `<video>` elements, whether a canvas gets tainted when the video is drawn on it.

The **stall timeout** is a user-agent defined length of time, which should be about three seconds. When a [media element](#) that is actively attempting to obtain [media data](#) has failed to receive any data for a duration equal to the [stall timeout](#), the user agent must [queue a task](#) to [fire a simple event](#) named `stalled` at the element.

User agents may allow users to selectively block or slow [media data](#) downloads. When a [media element](#)'s download has been blocked altogether, the user agent must act as if it was stalled (as opposed to acting as if the connection was closed). The rate of the download may also be throttled automatically by the user agent, e.g., to balance the download with other connections sharing the same bandwidth.

User agents may decide to not download more content at any time, e.g., after buffering five minutes of a one hour media resource, while waiting for the user to decide whether to play the resource or not, while waiting for user input in an interactive resource, or when the user navigates away from the page. When a [media element](#)'s download has been suspended, the user agent must [queue a task](#), to set the `networkState` to `NETWORK_IDLE` and [fire a simple event](#) named `suspend` at the element. If and when downloading of the resource resumes, the user agent must [queue a task](#) to set the `networkState` to `NETWORK_LOADING`. Between the queuing of these tasks, the load is suspended (so progress events don't

fire, as described above).

NOTE:

The `preload` attribute provides a hint regarding how much buffering the author thinks is advisable, even in the absence of the `autoplay` attribute.

When a user agent decides to completely suspend a download, e.g., if it is waiting until the user starts playback before downloading any further content, the user agent must [queue a task](#) to set the element's [delaying-the-load-event flag](#) to false. This stops [delaying the load event](#).

The user agent may use whatever means necessary to fetch the resource (within the constraints put forward by this and other specifications); for example, reconnecting to the server in the face of network errors, using HTTP range retrieval requests, or switching to a streaming protocol. The user agent must consider a resource erroneous only if it has given up trying to fetch it.

To determine the format of the [media resource](#), the user agent must use the [rules for sniffing audio and video specifically](#).

While the load is not suspended (see below), every 350ms (± 200 ms) or for every byte received, whichever is *least* frequent, [queue a task](#) to [fire a simple event](#) named `progress` at the element.

The [networking task source tasks](#) to process the data as it is being fetched must each [immediately queue a task](#) to run the first appropriate steps from the [media data processing steps list](#) below. (A new task is used for this so that the work described below occurs relative to the [media element event task source](#) rather than the [networking task source](#).)

When the [networking task source](#) has [queued](#) the last [task](#) as part of fetching the [media resource](#) (i.e., once the download has completed), if the fetching process completes without errors, including decoding the media data, and if all of the data is available to the user agent without network access, then, the user agent must move on to the *final step* below. This might never happen, e.g., when streaming an infinite resource such as Web radio, or if the resource is longer than the user agent's ability to cache data.

While the user agent might still need network access to obtain parts of the [media resource](#), the user agent must remain on this step.

EXAMPLE 415

For example, if the user agent has discarded the first half of a video, the user agent will remain at this step even once the [playback has ended](#), because there is always the chance the user will seek back to the start. In fact, in this situation, once [playback has ended](#), the user agent will end up firing a suspend event, as described earlier.

↪ Otherwise (*mode* is *local*)

The resource described by the [current media resource](#), if any, contains the [media data](#). It is [CORS-same-origin](#).

If the [current media resource](#) is a raw data stream (e.g., from a `File` object), then to determine the format of the [media resource](#), the user agent must use the [rules for sniffing audio and video specifically](#). Otherwise, if the data stream is pre-decoded, then the format is the format given by the relevant specification.

Whenever new data for the [current media resource](#) becomes available, [queue a task](#) to run the first appropriate steps from the [media data processing steps list](#) below.

When the [current media resource](#) is permanently exhausted (e.g., all the bytes of a `Blob` have been processed), if there were no decoding errors, then the user agent must move on to the *final step* below. This might never happen, e.g., if the [current media resource](#) is a `MediaStream`.

The [media data processing steps list](#) is as follows:

- ↪ If the [media data](#) cannot be fetched at all, due to network errors, causing the user agent to give up trying to fetch the resource
- ↪ If the [media data](#) can be fetched but is found by inspection to be in an unsupported format, or can otherwise not be rendered at all

DNS errors, HTTP 4xx and 5xx errors (and equivalents in other protocols), and other fatal network errors that occur before the user agent has established whether the *current media resource* is usable, as well as the file using an unsupported container format, or using unsupported codecs for all the data, must cause the user agent to execute the following steps:

1. The user agent should cancel the fetching process.
2. Abort this subalgorithm, returning to the [resource selection algorithm](#).

↩ If the [media resource](#) is found to have an audio track

1. Create an [AudioTrack](#) object to represent the audio track.
2. Update the [media element](#)'s [audioTracks](#) attribute's [AudioTrackList](#) object with the new [AudioTrack](#) object.
3. Let *enable* be *unknown*.
4. If either the [media resource](#) or the address of the *current media resource* indicate a particular set of audio tracks to enable, or if the user agent has information that would facilitate the selection of specific audio tracks to improve the user's experience, then: if this audio track is one of the ones to enable, then set *enable* to *true*, otherwise, set *enable* to *false*.

EXAMPLE 416

This could be triggered by *Media Fragments URI* fragment identifier syntax, but it could also be triggered e.g., by the user agent selecting a 5.1 surround sound audio track over a stereo audio track.

[\[MEDIA-FRAGS\]](#)

5. If *enable* is still *unknown*, then, if the [media element](#) does not yet have an enabled audio track, then set *enable* to *true*, otherwise, set *enable* to *false*.
6. If *enable* is *true*, then enable this audio track, otherwise, do not enable this audio track.
7. Fire a [trusted](#) event with the name `addtrack`, that does not bubble and is not cancelable, and that uses the [TrackEvent](#) interface, with the `track` attribute initialized to the new [AudioTrack](#) object, at this [AudioTrackList](#) object.

↩ If the [media resource](#) is found to have a video track

1. Create a [VideoTrack](#) object to represent the video track.
2. Update the [media element](#)'s [videoTracks](#) attribute's [VideoTrackList](#) object with the new [VideoTrack](#) object.
3. Let *enable* be *unknown*.
4. If either the [media resource](#) or the address of the *current media resource* indicate a particular set of video tracks to enable, or if the user agent has information that would facilitate the selection of specific video tracks to improve the user's experience, then: if this video track is the first such video track, then set *enable* to *true*, otherwise, set *enable* to *false*.

EXAMPLE 417

This could again be triggered by *Media Fragments URI* fragment identifier syntax.

5. If *enable* is still *unknown*, then, if the [media element](#) does not yet have a [selected](#) video track, then set *enable* to *true*, otherwise, set *enable* to *false*.
6. If *enable* is *true*, then select this track and unselect any previously selected video tracks, otherwise, do not select this video track. If other tracks are unselected, then a `change` event will be [fired](#).
7. Fire a [trusted](#) event with the name `addtrack`, that does not bubble and is not cancelable, and that uses the [TrackEvent](#) interface, with the `track` attribute initialized to the new [VideoTrack](#) object, at this [VideoTrackList](#) object.

↩ Once enough of the [media data](#) has been fetched to determine the duration of the [media resource](#), its dimensions, and other metadata

This indicates that the resource is usable. The user agent must follow these substeps:

1. Establish the [media timeline](#) for the purposes of the [current playback position](#) and the [earliest possible position](#), based on the [media data](#).
2. Update the [timeline offset](#) to the date and time that corresponds to the zero time in the [media timeline](#) established in the previous step, if any. If no explicit time and date is given by the [media resource](#), the [timeline offset](#) must be set to Not-a-Number (NaN).
3. Set the [current playback position](#) and the [official playback position](#) to the [earliest possible position](#).
4. Update the duration attribute with the time of the last frame of the resource, if known, on the [media timeline](#) established above. If it is not known (e.g., a stream that is in principle infinite), update the duration attribute to the value positive Infinity.

NOTE:

The user agent will [queue a task](#) to [fire a simple event](#) named `durationchange` at the element at this point.

5. For `<video>` elements, set the `videoWidth` and `videoHeight` attributes, and [queue a task](#) to [fire a simple event](#) named `resize` at the [media element](#).

NOTE:

Further `resize` events will be fired if the dimensions subsequently change.

6. Set the `readyState` attribute to `HAVE_METADATA`.

NOTE:

A `loadedmetadata` DOM event will be [fired](#) as part of setting the `readyState` attribute to a new value.

7. Let *jumped* be false.
8. If the [media element](#)'s [default playback start position](#) is greater than zero, then [seek](#) to that time, and let *jumped* be true.
9. Let the [media element](#)'s [default playback start position](#) be zero.
10. Let the *initial playback position* be zero.
11. If either the [media resource](#) or the address of the [current media resource](#) indicate a particular start time, then set the *initial playback position* to that time and, if *jumped* is still false, [seek](#) to that time and let *jumped* be true.

EXAMPLE 418

For example, with media formats that support the *Media Fragments URI* fragment identifier syntax, the fragment identifier can be used to indicate a start position. [\[MEDIA-FRAGS\]](#)

12. If there is no enabled audio track, then enable an audio track. This will cause a `change` event to be [fired](#).
13. If there is no [selected](#) video track, then select a video track. This will cause a `change` event to be [fired](#).

Once the `readyState` attribute reaches `HAVE_CURRENT_DATA`, after the `loadeddata` event has been [fired](#), set the element's [delaying-the-load-event flag](#) to false. This stops [delaying the load event](#).

NOTE:

A user agent that is attempting to reduce network usage while still fetching the metadata for each [media resource](#) would also stop buffering at this point, following [the rules described previously](#), which involve the `networkState` attribute switching to the `NETWORK_IDLE` value and a `suspend` event firing.

NOTE:

The user agent is required to determine the duration of the [media resource](#) and go through this step before playing.

↪ Once the entire [media resource](#) has been fetched (but potentially before any of it has been decoded)

[Fire a simple event](#) named progress at the [media element](#).

Set the `networkState` to `NETWORK_IDLE` and [fire a simple event](#) named suspend at the [media element](#).

If the user agent ever discards any [media data](#) and then needs to resume the network activity to obtain it again, then it must [queue a task](#) to set the `networkState` to `NETWORK_LOADING`.

NOTE:

If the user agent can keep the [media resource](#) loaded, then the algorithm will continue to its final step below, which aborts the algorithm.

↪ If the connection is interrupted after some [media data](#) has been received, causing the user agent to give up trying to fetch the resource

Fatal network errors that occur after the user agent has established whether the *current media resource* is usable (i.e., once the [media element](#)'s `readyState` attribute is no longer `HAVE_NOTHING`) must cause the user agent to execute the following steps:

1. The user agent should cancel the fetching process.
2. Set the error attribute to a new `MediaError` object whose `code` attribute is set to `MEDIA_ERR_NETWORK`.
3. Set the element's `networkState` attribute to the `NETWORK_IDLE` value.
4. Set the element's [delaying-the-load-event flag](#) to false. This stops [delaying the load event](#).
5. [Fire a simple event](#) named error at the [media element](#).
6. Abort the overall [resource selection algorithm](#).

⚠↪ If the [media data](#) is corrupted

Fatal errors in decoding the [media data](#) that occur after the user agent has established whether the *current media resource* is usable (i.e., once the [media element](#)'s `readyState` attribute is no longer `HAVE_NOTHING`) must cause the user agent to execute the following steps:

1. The user agent should cancel the fetching process.
2. Set the error attribute to a new `MediaError` object whose `code` attribute is set to `MEDIA_ERR_DECODE`.
3. Set the element's `networkState` attribute to the `NETWORK_IDLE` value.
4. Set the element's [delaying-the-load-event flag](#) to false. This stops [delaying the load event](#).
5. [Fire a simple event](#) named error at the [media element](#).
6. Abort the overall [resource selection algorithm](#).

↪ If the [media data](#) fetching process is aborted by the user

The fetching process is aborted by the user, e.g., because the user pressed a "stop" button, the user agent must execute the following steps. These steps are not followed if the `load()` method itself is invoked while these steps are running, as the steps above handle that particular kind of abort.

1. The user agent should cancel the fetching process.
2. Set the error attribute to a new `MediaError` object whose `code` attribute is set to `MEDIA_ERR_ABORTED`.
3. [Fire a simple event](#) named abort at the [media element](#).
4. If the [media element](#)'s `readyState` attribute has a value equal to `HAVE_NOTHING`, set the element's `networkState` attribute to the `NETWORK_EMPTY` value, set the element's [show poster flag](#) to true, and [fire](#)

a [simple event](#) named [emptied](#) at the element.

Otherwise, set the element's `networkState` attribute to the `NETWORK_IDLE` value.

5. Set the element's [delaying-the-load-event flag](#) to false. This stops [delaying the load event](#).
6. Abort the overall [resource selection algorithm](#).

❧ If the [media data](#) can be fetched but has non-fatal errors or uses, in part, codecs that are unsupported, preventing the user agent from rendering the content completely correctly but not preventing playback altogether

The server returning data that is partially usable but cannot be optimally rendered must cause the user agent to render just the bits it can handle, and ignore the rest.

❧ If the [media resource](#) is found to declare a [media-resource-specific text track](#) that the user agent supports

If the [media data](#) is [CORS-same-origin](#), run the [steps to expose a media-resource-specific text track](#) with the relevant data.

NOTE:

Cross-origin videos do not expose their subtitles, since that would allow attacks such as hostile sites reading subtitles from confidential videos on a user's intranet.

5. *Final step*: If the user agent ever reaches this step (which can only happen if the entire resource gets loaded and kept available); abort the overall [resource selection algorithm](#).

When a [media element](#) is to **forget the media element's media-resource-specific tracks**, the user agent must remove from the [media element's list of text tracks](#) all the [media-resource-specific text tracks](#), then empty the [media element's](#) `audioTracks` attribute's `AudioTrackList` object, then empty the [media element's](#) `videoTracks` attribute's [VideoTrackList](#) object. No events (in particular, no `removetrack` events) are fired as part of this; the error and [emptied](#) events, fired by the algorithms that invoke this one, can be used instead.



The `preload` attribute is an [enumerated attribute](#). The following table lists the keywords and states for the attribute — the keywords in the left column map to the states in the cell in the second column on the same row as the keyword. The attribute can be changed even once the [media resource](#) is being buffered or played; the descriptions in the table below are to be interpreted with that in mind.

Keyword	State	Brief description
'none'	None	Hints to the user agent that either the author does not expect the user to need the media resource, or that the server wants to minimize unnecessary traffic. This state does not provide a hint regarding how aggressively to actually download the media resource if buffering starts anyway (e.g., once the user hits "play").
'metadata'	Metadata	Hints to the user agent that the author does not expect the user to need the media resource, but that fetching the resource metadata (dimensions, track list, duration, etc), and maybe even the first few frames, is reasonable. If the user agent precisely fetches no more than the metadata, then the media element will end up with its <code>readyState</code> attribute set to <code>HAVE_METADATA</code> ; typically though, some frames will be obtained as well and it will probably be <code>HAVE_CURRENT_DATA</code> or <code>HAVE_FUTURE_DATA</code> . When the media resource is playing, hints to the user agent that bandwidth is to be considered scarce, e.g., suggesting throttling the download so that the media data is obtained at the slowest possible rate that still maintains consistent playback.
'auto'	Automatic	Hints to the user agent that the user agent can put the user's needs first without risk to the server, up to and including optimistically downloading the entire resource.

The empty string is also a valid keyword, and maps to the [Automatic](#) state. The attribute's *missing value default* is user-agent defined, though the [Metadata](#) state is suggested as a compromise between reducing server load and providing an optimal user experience.

NOTE:

Authors might switch the attribute from "none" or "metadata" to "auto" dynamically once the user begins playback. For example, on a page with many videos this might be used to indicate that the many videos are not to be downloaded unless requested, but that once one is requested it is to be downloaded aggressively.

The `preload` attribute is intended to provide a hint to the user agent about what the author thinks will lead to the best user experience. The attribute may be ignored altogether, for example based on explicit user preferences or based on the available connectivity.

The `preload` IDL attribute must [reflect](#) the content attribute of the same name, [limited to only known values](#).

NOTE:

The `autoplay` attribute can override the `preload` attribute (since if the media plays, it naturally has to buffer first, regardless of the hint given by the `preload` attribute). Including both is not an error, however.



This definition is non-normative. Implementation requirements are given below this definition.

`media . buffered`

Returns a `TimeRanges` object that represents the ranges of the [media resource](#) that the user agent has buffered.

The `buffered` attribute must return a new static [normalized TimeRanges object](#) that represents the ranges of the [media resource](#), if any, that the user agent has buffered, at the time the attribute is evaluated. Users agents must accurately determine the ranges available, even for media streams where this can only be determined by tedious inspection.

NOTE:

Typically this will be a single range anchored at the zero point, but if, e.g., the user agent uses HTTP range requests in response to seeking, then there could be multiple ranges.

User agents may discard previously buffered data.

NOTE:

Thus, a time position included within a range of the objects return by the `buffered` attribute at one time can end up being not included in the range(s) of objects returned by the same attribute at later times.

4.7.14.6. Offsets into the media resource

This definition is non-normative. Implementation requirements are given below this definition.

`media . duration`

Returns the length of the [media resource](#), in seconds, assuming that the start of the [media resource](#) is at time zero.

Returns NaN if the duration isn't available.

Returns Infinity for unbounded streams.

`media . currentTime [ = value ]`

Returns the [official playback position](#), in seconds.

Can be set, to seek to the given time.

A [media resource](#) has a **media timeline** that maps times (in seconds) to positions in the [media resource](#). The origin of a timeline is its earliest defined position. The duration of a timeline is its last defined position.

Establishing the media timeline: If the [media resource](#) somehow specifies an explicit timeline whose origin is not negative (i.e., gives each frame a specific time offset and gives the first frame a zero or positive offset), then the [media timeline](#) should be that timeline. (Whether the [media resource](#) can specify a timeline or not depends on the [media resource's](#) format.)

If the [media resource](#) specifies an explicit start time *and date*, then that time and date should be considered the zero point in the [media timeline](#); the [timeline offset](#) will be the time and date, exposed using the `getStartDate()` method.

If the [media resource](#) has a discontinuous timeline, the user agent must extend the timeline used at the start of the resource across the entire resource, so that the [media timeline](#) of the [media resource](#) increases linearly starting from the [earliest possible position](#) (as defined below), even if the underlying [media data](#) has out-of-order or even overlapping time codes.

EXAMPLE 419

For example, if two clips have been concatenated into one video file, but the video format exposes the original times for the two clips, the video data might expose a timeline that goes, say, 00:15..00:29 and then 00:05..00:38. However, the user agent would not expose those times; it would instead expose the times as 00:15..00:29 and 00:29..01:02, as a single video.

In the rare case of a [media resource](#) that does not have an explicit timeline, the zero time on the [media timeline](#) should correspond to the first frame of the [media resource](#). In the even rarer case of a [media resource](#) with no explicit timings of any kind, not even frame durations, the user agent must itself determine the time for each frame in a user-agent-defined manner.



NOTE:

An example of a file format with no explicit timeline but with explicit frame durations is the Animated GIF format. An example of a file format with no explicit timings at all is the JPEG-push format (multipart/x-mixed-replace with JPEG frames, often used as the format for MJPEG streams).

If, in the case of a resource with no timing information, the user agent will nonetheless be able to seek to an earlier point than the first frame originally provided by the server, then the zero time should correspond to the earliest seekable time of the [media resource](#); otherwise, it should correspond to the first frame received from the server (the point in the [media resource](#) at which the user agent began receiving the stream).

NOTE:

At the time of writing, there is no known format that lacks explicit frame time offsets yet still supports seeking to a frame before the first frame sent by the server.

EXAMPLE 420

Consider a stream from a TV broadcaster, which begins streaming on a sunny Friday afternoon in October, and always sends connecting user agents the media data on the same media timeline, with its zero time set to the start of this stream. Months later, user agents connecting to this stream will find that the first frame they receive has a time with millions of seconds. The `getStartDate()` method would always return the date that the broadcast started; this would allow controllers to display real times in their scrubber (e.g., "2:30pm") rather than a time relative to when the broadcast began ("8 months, 4 hours, 12 minutes, and 23 seconds").

Consider a stream that carries a video with several concatenated fragments, broadcast by a server that does not allow user agents to request specific times but instead just streams the video data in a predetermined order, with the first frame delivered always being identified as the frame with time zero. If a user agent connects to this stream and receives fragments defined as covering timestamps 2010-03-20 23:15:00 UTC to 2010-03-21 00:05:00 UTC and 2010-02-12 14:25:00 UTC to 2010-02-12 14:35:00 UTC, it would expose this with a [media timeline](#) starting at 0s and extending to 3,600s (one hour). Assuming the streaming server disconnected at the end of the second clip, the `duration` attribute would then return 3,600. The `getStartDate()` method would return a [Date](#) object with a time corresponding to 2010-03-20 23:15:00 UTC. However, if a different user agent connected five minutes later, it would (presumably) receive fragments covering timestamps 2010-03-20 23:20:00 UTC to 2010-03-21 00:05:00 UTC and 2010-02-12 14:25:00 UTC to 2010-02-12 14:35:00 UTC, and would expose this with a [media timeline](#) starting at 0s and extending to 3,300s (fifty five minutes). In this case, the `getStartDate()` method would return a [Date](#) object with a time corresponding to 2010-03-20 23:20:00 UTC.

In both of these examples, the `seekable` attribute would give the ranges that the controller would want to actually display in its UI; typically, if the servers don't support seeking to arbitrary times, this would be the range of time from the moment the user agent connected to the stream up to the latest frame that the user agent has obtained; however, if the user agent starts discarding earlier information, the actual range might be shorter.

In any case, the user agent must ensure that the [earliest possible position](#) (as defined below) using the established [media timeline](#), is greater than or equal to zero.

The [media timeline](#) also has an associated clock. Which clock is used is user-agent defined, and may be [media resource](#)-dependent, but it should approximate the user's wall clock.

[Media elements](#) have a **current playback position**, which must initially (i.e., in the absence of [media data](#)) be zero seconds. The [current playback position](#) is a time on the [media timeline](#).

[Media elements](#) also have an **official playback position**, which must initially be set to zero seconds. The [official playback position](#) is an approximation of the [current playback position](#) that is kept stable while scripts are running.

[Media elements](#) also have a **default playback start position**, which must initially be set to zero seconds. This time is used to allow the element to be seeked even before the media is loaded.

Each [media element](#) has a **show poster flag**. When a [media element](#) is created, this flag must be set to true. This flag is used to control when the user agent is to show a poster frame for a `<video>` element instead of showing the video contents.

The `currentTime` attribute must, on getting, return the [media element's default playback start position](#), unless that is zero, in which case it must return the element's [official playback position](#). The returned value must be expressed in seconds. On setting, if the [media element's](#) `readyState` is `HAVE_NOTHING`, then it must set the [media element's default playback start position](#) to the new value; otherwise, it must set the [official playback position](#) to the new value and then [seek](#) to the new value. The new value must be interpreted as being in seconds.

[Media elements](#) have an **initial playback position**, which must initially (i.e., in the absence of [media data](#)) be zero seconds. The [initial playback position](#) is updated when a [media resource](#) is loaded. The [initial playback position](#) is a time on the [media timeline](#).

If the [media resource](#) is a streaming resource, then the user agent might be unable to obtain certain parts of the resource after it has expired from its buffer. Similarly, some [media resources](#) might have a [media timeline](#) that doesn't start at zero. The **earliest possible position** is the earliest position in the stream or resource that the user agent can ever obtain again. It is also a time on the [media timeline](#).

NOTE:

The [earliest possible position](#) is not explicitly exposed in the API; it corresponds to the start time of the first range in the seekable attribute's `TimeRanges` object, if any, or the [current playback position](#) otherwise.

When the [earliest possible position](#) changes, then: if the [current playback position](#) is before the [earliest possible position](#), the user agent must [seek](#) to the [earliest possible position](#); otherwise, if the user agent has not fired a `timeupdate` event at the element in the past 15 to 250ms and is not still running event handlers for such an event, then the user agent must [queue a task](#) to [fire a simple event](#) named `timeupdate` at the element.

NOTE:

Because of the above requirement and the requirement in the [resource fetch algorithm](#) that kicks in [when the metadata of the clip becomes known](#), the [current playback position](#) can never be less than the [earliest possible position](#).

If at any time the user agent learns that an audio or video track has ended and all [media data](#) relating to that track corresponds to parts of the [media timeline](#) that are *before* the [earliest possible position](#), the user agent may [queue a task](#) to first remove the track from the `audioTracks` attribute's `AudioTrackList` object or the `videoTracks` attribute's `VideoTrackList` object as appropriate and then [fire](#) a [trusted](#) event with the name `removetrack`, that does not bubble and is not cancelable, and that uses the [TrackEvent](#) interface, with the `track` attribute initialized to the `AudioTrack` or `VideoTrack` object representing the track, at the [media element's](#) aforementioned `AudioTrackList` or `VideoTrackList` object.

The **duration** attribute must return the time of the end of the [media resource](#), in seconds, on the [media timeline](#). If no [media data](#) is available, then the attributes must return the Not-a-Number (NaN) value. If the [media resource](#) is not known to be bounded (e.g., streaming radio, or a live event with no announced end time), then the attribute must return the positive Infinity value.

The user agent must determine the duration of the [media resource](#) before playing any part of the [media data](#) and before setting `readyState` to a value equal to or greater than `HAVE_METADATA`, even if doing so requires fetching multiple parts of the resource.

When the length of the [media resource](#) changes to a known value (e.g., from being unknown to known, or from a previously established length to a new length) the user agent must [queue a task](#) to [fire a simple event](#) named [durationchange](#) at the [media element](#). (The event is not fired when the duration is reset as part of loading a new media resource.) If the duration is changed such that the [current playback position](#) ends up being greater than the time of the end of the [media resource](#), then the user agent must also [seek](#) to the time of the end of the [media resource](#).

EXAMPLE 421

If an "infinite" stream ends for some reason, then the duration would change from positive Infinity to the time of the last frame or sample in the stream, and the [durationchange](#) event would be fired. Similarly, if the user agent initially estimated the [media resource](#)'s duration instead of determining it precisely, and later revises the estimate based on new information, then the duration would change and the [durationchange](#) event would be fired.

Some video files also have an explicit date and time corresponding to the zero time in the [media timeline](#), known as the [timeline offset](#). Initially, the [timeline offset](#) must be set to Not-a-Number (NaN).

The [getStartDate\(\)](#) method must return [a new Date object](#) representing the current [timeline offset](#).



The **loop** attribute is a [boolean attribute](#) that, if specified, indicates that the [media element](#) is to seek back to the start of the [media resource](#) upon reaching the end.

The **loop** IDL attribute must [reflect](#) the content attribute of the same name.

4.7.14.7. Ready states

This definition is non-normative. Implementation requirements are given below this definition.

media . readyState

Returns a value that expresses the current state of the element with respect to rendering the [current playback position](#), from the codes in the list below.

[Media elements](#) have a *ready state*, which describes to what degree they are ready to be rendered at the [current playback position](#). The possible values are as follows; the ready state of a media element at any particular time is the greatest value describing the state of the element:

HAVE_NOTHING (numeric value 0)

No information regarding the [media resource](#) is available. No data for the [current playback position](#) is available. [Media elements](#) whose [networkState](#) attribute are set to NETWORK_EMPTY are always in the HAVE_NOTHING state.

HAVE_METADATA (numeric value 1)

Enough of the resource has been obtained that the duration of the resource is available. In the case of a `<video>` element, the dimensions of the video are also available. No [media data](#) is available for the immediate [current playback position](#).

HAVE_CURRENT_DATA (numeric value 2)

Data for the immediate [current playback position](#) is available, but either not enough data is available that the user agent could successfully advance the [current playback position](#) in the [direction of playback](#) at all without immediately reverting to the HAVE_METADATA state, or there is no more data to obtain in the [direction of playback](#). For example, in video this corresponds to the user agent having data from the current frame, but not the next frame, when the [current playback position](#) is at the end of the current frame; and to when [playback has ended](#).

HAVE_FUTURE_DATA (numeric value 3)

Data for the immediate [current playback position](#) is available, as well as enough data for the user agent to advance the [current playback position](#) in the [direction of playback](#) at least a little without immediately reverting to the HAVE_METADATA state, and [the text tracks are ready](#). For example, in video this corresponds to the user agent having data for at least the current frame and the next frame when the [current playback position](#) is at the instant in time between the two frames, or to the user agent having the video data for the current frame and audio data to keep playing at least a little when the [current playback position](#) is in the middle of a frame. The user agent cannot be in this state if

[playback has ended](#), as the [current playback position](#) can never advance in this case.

HAVE_ENOUGH_DATA (numeric value 4)

All the conditions described for the HAVE_FUTURE_DATA state are met, and, in addition, either of the following conditions is also true:

- The user agent estimates that data is being fetched at a rate where the [current playback position](#), if it were to advance at the [effective playback rate](#), would not overtake the available data before playback reaches the end of the [media resource](#).
- The user agent has entered a state where waiting longer will not result in further data being obtained, and therefore nothing would be gained by delaying playback any further. (For example, the buffer might be full.)

NOTE:

In practice, the difference between HAVE_METADATA and HAVE_CURRENT_DATA is negligible. Really the only time the difference is relevant is when painting a `<video>` element onto a `<canvas>`, where it distinguishes the case where something will be drawn (HAVE_CURRENT_DATA or greater) from the case where nothing is drawn (HAVE_METADATA or less). Similarly, the difference between HAVE_CURRENT_DATA (only the current frame) and HAVE_FUTURE_DATA (at least this frame and the next) can be negligible (in the extreme, only one frame). The only time that distinction really matters is when a page provides an interface for "frame-by-frame" navigation.

When the ready state of a [media element](#) whose networkState is not NETWORK_EMPTY changes, the user agent must follow the steps given below:

1. Apply the first applicable set of substeps from the following list:

↪ If the previous ready state was HAVE_NOTHING, and the new ready state is HAVE_METADATA

[Queue a task](#) to [fire a simple event](#) named loadedmetadata at the element.

NOTE:

Before this task is run, as part of the [event loop](#) mechanism, the rendering will have been updated to re-size the `<video>` element if appropriate.

⏸↪ If the previous ready state was HAVE_METADATA and the new ready state is HAVE_CURRENT_DATA or greater

If this is the first time this occurs for this [media element](#) since the load() algorithm was last invoked, the user agent must [queue a task](#) to [fire a simple event](#) named loadeddata at the element.

If the new ready state is HAVE_FUTURE_DATA or HAVE_ENOUGH_DATA, then the relevant steps below must then be run also.

↪ If the previous ready state was HAVE_FUTURE_DATA or more, and the new ready state is HAVE_CURRENT_DATA or less

If the [media element](#) was [potentially playing](#) before its readyState attribute changed to a value lower than HAVE_FUTURE_DATA, and the element has not [ended playback](#), and playback has not [stopped due to errors](#), [paused for user interaction](#), or [paused for in-band content](#), the user agent must [queue a task](#) to [fire a simple event](#) named timeupdate at the element, and [queue a task](#) to [fire a simple event](#) named waiting at the element.

↪ If the previous ready state was HAVE_CURRENT_DATA or less, and the new ready state is HAVE_FUTURE_DATA

The user agent must [queue a task](#) to [fire a simple event](#) named canplay at the element.

If the element's paused attribute is false, the user agent must [queue a task](#) to [fire a simple event](#) named playing at the element.

↪ If the new ready state is HAVE_ENOUGH_DATA

If the previous ready state was HAVE_CURRENT_DATA or less, the user agent must [queue a task](#) to [fire a simple event](#) named canplay at the element, and, if the element's paused attribute is false, [queue a task](#) to [fire a simple event](#) named playing at the element.

If the [autoplaying flag](#) is true, and the paused attribute is true, and the [media element](#) has an autoplay attribute specified, and the [media element](#)'s [node document](#)'s [active sandboxing flag set](#) does not have the

[sandboxed automatic features browsing context flag](#) set, then the user agent may also run the following sub-steps:

1. Set the `paused` attribute to false.
2. If the element's `show poster flag` is true, set it to false and run the *time marches on* steps.
3. [Queue a task](#) to [fire a simple event](#) named `play` at the element.
4. [Queue a task](#) to [fire a simple event](#) named `playing` at the element.
5. Set the `autoplaying flag` to false.

NOTE:

User agents do not need to support autoplay, and it is suggested that user agents honor user preferences on the matter. Authors are urged to use the `autoplay` attribute rather than using script to force the video to play, so as to allow the user to override the behavior if so desired.

In any case, the user agent must finally [queue a task](#) to [fire a simple event](#) named `canplaythrough` at the element.

NOTE:

It is possible for the ready state of a media element to jump between these states discontinuously. For example, the state of a media element can jump straight from `HAVE_METADATA` to `HAVE_ENOUGH_DATA` without passing through the `HAVE_CURRENT_DATA` and `HAVE_FUTURE_DATA` states.

The **readyState** IDL attribute must, on getting, return the value described above that describes the current ready state of the [media element](#).

The **autoplay** attribute is a [boolean attribute](#). When present, the user agent (as described in the algorithm described herein) will automatically begin playback of the [media resource](#) as soon as it can do so without stopping.

NOTE:

Authors are urged to use the `autoplay` attribute rather than using script to trigger automatic playback, as this allows the user to override the automatic playback when it is not desired, e.g., when using a screen reader. Authors are also encouraged to consider not using the automatic playback behavior at all, and instead to let the user agent wait for the user to start playback explicitly.

The **autoplay** IDL attribute must [reflect](#) the content attribute of the same name.

4.7.14.8. Playing the media resource

This definition is non-normative. Implementation requirements are given below this definition.

`media` . `paused`

Returns true if playback is paused; false otherwise.

`media` . `ended`

Returns true if playback has reached the end of the [media resource](#).

`media` . `defaultPlaybackRate` [= *value*]

Returns the default rate of playback, for when the user is not fast-forwarding or reversing through the [media resource](#).

Can be set, to change the default rate of playback.

The default rate has no direct effect on playback, but if the user switches to a fast-forward mode, when they return to the normal playback mode, it is expected that the rate of playback will be returned to the default rate of playback.

`media` . `playbackRate` [= *value*]

Returns the current rate playback, where 1.0 is normal speed.

Can be set, to change the rate of playback.

`media . played`

Returns a `TimeRanges` object that represents the ranges of the [media resource](#) that the user agent has played.

`media . play()`

Sets the `paused` attribute to false, loading the [media resource](#) and beginning playback if necessary. If the playback had ended, will restart it from the start.

`media . pause()`

Sets the `paused` attribute to true, loading the [media resource](#) if necessary.

The **paused** attribute represents whether the [media element](#) is paused or not. The attribute must initially be true.

A [media element](#) is a **blocked media element** if its `readyState` attribute is in the `HAVE_NOTHING` state, the `HAVE_METADATA` state, or the `HAVE_CURRENT_DATA` state, or if the element has [paused for user interaction](#) or [paused for in-band content](#).

A [media element](#) is said to be **potentially playing** when its `paused` attribute is false, the element has not [ended playback](#), playback has not [stopped due to errors](#), and the element is not a [blocked media element](#).

NOTE:

A waiting DOM event can be [fired](#) as a result of an element that is [potentially playing](#) stopping playback due to its `readyState` attribute changing to a value lower than `HAVE_FUTURE_DATA`.

A [media element](#) is said to have **ended playback** when:

- The element's `readyState` attribute is `HAVE_METADATA` or greater, and
- Either:
 - The [current playback position](#) is the end of the [media resource](#), and
 - The [direction of playback](#) is forwards, and
 - The [media element](#) does not have a `loop` attribute specified.

Or:

- The [current playback position](#) is the [earliest possible position](#), and
- The [direction of playback](#) is backwards.

The **ended** attribute must return true if, the last time the [event loop](#) reached step 1, the [media element](#) had [ended playback](#) and the [direction of playback](#) was forwards, and false otherwise.

A [media element](#) is said to have **stopped due to errors** when the element's `readyState` attribute is `HAVE_METADATA` or greater, and the user agent [encounters a non-fatal error](#) during the processing of the [media data](#), and due to that error, is not able to play the content at the [current playback position](#).

A [media element](#) is said to have **paused for user interaction** when its `paused` attribute is false, the `readyState` attribute is either `HAVE_FUTURE_DATA` or `HAVE_ENOUGH_DATA` and the user agent has reached a point in the [media resource](#) where the user has to make a selection for the resource to continue.

It is possible for a [media element](#) to have both [ended playback](#) and [paused for user interaction](#) at the same time.

When a [media element](#) that is [potentially playing](#) stops playing because it has [paused for user interaction](#), the user agent must [queue a task](#) to [fire a simple event](#) named `timeupdate` at the element.

A [media element](#) is said to have **paused for in-band content** when its `paused` attribute is false, the `readyState` attribute is either `HAVE_FUTURE_DATA` or `HAVE_ENOUGH_DATA` and the user agent has suspended playback of the [media resource](#) in order to play content that is temporally anchored to the [media resource](#) and has a non-zero length, or to play content that is temporally anchored to a segment of the [media resource](#) but has a length longer than that segment.

EXAMPLE 422

One example of when a [media element](#) would be [paused for in-band content](#) is when the user agent is playing [audio descriptions](#) from an external WebVTT file, and the synthesized speech generated for a cue is longer than the time between the [text track cue start time](#) and the [text track cue end time](#).



When the [current playback position](#) reaches the end of the [media resource](#) when the [direction of playback](#) is forwards, then the user agent must follow these steps:

1. If the [media element](#) has a loop attribute specified, then [seek](#) to the [earliest possible position](#) of the [media resource](#) and abort these steps.
2. As defined above, the ended IDL attribute starts returning true once the [event loop](#) returns to step 1.
3. [Queue a task](#) to [fire a simple event](#) named `timeupdate` at the [media element](#).
4. [Queue a task](#) that, if the [media element](#) has still [ended playback](#), and the [direction of playback](#) is still forwards, and [paused](#) is false, changes [paused](#) to true and [fires a simple event](#) named `pause` at the [media element](#).
5. [Queue a task](#) to [fire a simple event](#) named `ended` at the [media element](#).

When the [current playback position](#) reaches the [earliest possible position](#) of the [media resource](#) when the [direction of playback](#) is backwards, then the user agent must only [queue a task](#) to [fire a simple event](#) named `timeupdate` at the element.

NOTE:

The word "reaches" here does not imply that the [current playback position](#) needs to have changed during normal playback; it could be via [seeking](#), for instance.



The `defaultPlaybackRate` attribute gives the desired speed at which the [media resource](#) is to play, as a multiple of its intrinsic speed. The attribute is mutable: on getting it must return the last value it was set to, or 1.0 if it hasn't yet been set; on setting the attribute must be set to the new value.

NOTE:

The `defaultPlaybackRate` is used by the user agent when it [exposes a user interface to the user](#).

The `playbackRate` attribute gives the [effective playback rate](#) which is the speed at which the [media resource](#) plays, as a multiple of its intrinsic speed. If it is not equal to the `defaultPlaybackRate`, then the implication is that the user is using a feature such as fast forward or slow motion playback. The attribute is mutable: on getting it must return the last value it was set to, or 1.0 if it hasn't yet been set; on setting the attribute must be set to the new value, and the playback will change speed (if the element is [potentially playing](#)).

When the `defaultPlaybackRate` or `playbackRate` attributes change value (either by being set by script or by being changed directly by the user agent, e.g., in response to user control) the user agent must [queue a task](#) to [fire a simple event](#) named `ratechange` at the [media element](#).



The `played` attribute must return a new static [normalized TimeRanges object](#) that represents the ranges of points on the [media timeline](#) of the [media resource](#) reached through the usual monotonic increase of the [current playback position](#) during normal playback, if any, at the time the attribute is evaluated.



When the `play()` method on a [media element](#) is invoked, the user agent must run the following steps.

1. If the [media element](#)'s `networkState` attribute has the value `NETWORK_EMPTY`, invoke the [media element](#)'s [resource selection algorithm](#).
2. If the [playback has ended](#) and the [direction of playback](#) is forwards, [seek](#) to the [earliest possible position](#) of the [media resource](#).

NOTE:

This will cause the user agent to [queue a task](#) to [fire a simple event](#) named `timeupdate` at the [media element](#).

3. If the [media element](#)'s `paused` attribute is true, run the following substeps:
 1. Change the value of `paused` to false.
 2. If the [show poster flag](#) is true, set the element's [show poster flag](#) to false and run the *time marches on* steps.
 3. [Queue a task](#) to [fire a simple event](#) named `play` at the element.
 4. If the [media element](#)'s `readyState` attribute has the value `HAVE_NOTHING`, `HAVE_METADATA`, or `HAVE_CURRENT_DATA`, [queue a task](#) to [fire a simple event](#) named `waiting` at the element.

Otherwise, the [media element](#)'s `readyState` attribute has the value `HAVE_FUTURE_DATA` or `HAVE_ENOUGH_DATA`: [queue a task](#) to [fire a simple event](#) named `playing` at the element.
4. Set the [media element](#)'s [autoplaying flag](#) to false.



When the **'pause()'** method is invoked, and when the user agent is required to pause the [media element](#), the user agent must run the following steps:

1. If the [media element](#)'s `networkState` attribute has the value `NETWORK_EMPTY`, invoke the [media element](#)'s [resource selection algorithm](#).
2. Run the [internal pause steps](#) for the [media element](#).

The **internal pause steps** for a [media element](#) are as follows:

1. Set the [media element](#)'s [autoplaying flag](#) to false.
2. If the [media element](#)'s `paused` attribute is false, run the following steps:
 1. Change the value of `paused` to true.
 2. [Queue a task](#) to [fire a simple event](#) named `timeupdate` at the element.
 3. [Queue a task](#) to [fire a simple event](#) named `pause` at the element.
 4. Set the [official playback position](#) to the [current playback position](#).



The **effective playback rate** is just the element's `playbackRate`.

If the [effective playback rate](#) is positive or zero, then the **direction of playback** is forwards. Otherwise, it is backwards.

When a [media element](#) is [potentially playing](#) and its [Document](#) is a [fully active Document](#), its [current playback position](#) must increase monotonically at [effective playback rate](#) units of media time per unit time of the [media timeline](#)'s clock. (This specification always refers to this as an *increase*, but that increase could actually be a *decrease* if the [effective playback rate](#) is negative.)

NOTE:

The [effective playback rate](#) can be 0.0, in which case the [current playback position](#) doesn't move, despite playback not being paused (`paused` doesn't become true, and the `pause` event doesn't fire).

NOTE:

This specification doesn't define how the user agent achieves the appropriate playback rate — depending on the protocol and media available, it is plausible that the user agent could negotiate with the server to have the server provide the media data at the appropriate rate, so that (except for the period between when the rate is changed and when the server updates the stream's playback rate) the client doesn't actually have to drop or interpolate any frames.

Any time the user agent [provides a stable state](#), the [official playback position](#) must be set to the [current playback position](#).

While the [direction of playback](#) is backwards, any corresponding audio must be [muted](#). While the [effective playback rate](#) is so low or so high that the user agent cannot play audio usefully, the corresponding audio must also be [muted](#). If the [effective playback rate](#) is not 1.0, the user agent may apply pitch adjustments to the audio as necessary to render it faithfully.

[Media elements](#) that are [potentially playing](#) while not [in a Document](#) must not play any video, but should play any audio component. Media elements must not stop playing just because all references to them have been removed; only once a media element is in a state where no further audio could ever be played by that element may the element be garbage collected.

NOTE:

It is possible for an element to which no explicit references exist to play audio, even if such an element is not still actively playing: for instance, a [media element](#) whose [media resource](#) has no audio tracks could eventually play audio again if it had an event listener that changes the [media resource](#).



Each [media element](#) has a **list of newly introduced cues**, which must be initially empty. Whenever a [text track cue](#) is added to the [list of cues](#) of a [text track](#) that is in the [list of text tracks](#) for a [media element](#), that [cue](#) must be added to the [media element's list of newly introduced cues](#). Whenever a [text track](#) is added to the [list of text tracks](#) for a [media element](#), all of the [cues](#) in that [text track's list of cues](#) must be added to the [media element's list of newly introduced cues](#). When a [media element's list of newly introduced cues](#) has new cues added while the [media element's show poster flag](#) is not set, then the user agent must run the *time marches on* steps.

When a [text track cue](#) is removed from the [list of cues](#) of a [text track](#) that is in the [list of text tracks](#) for a [media element](#), and whenever a [text track](#) is removed from the [list of text tracks](#) of a [media element](#), if the [media element's show poster flag](#) is not set, then the user agent must run the *time marches on* steps.

When the [current playback position](#) of a [media element](#) changes (e.g., due to playback or seeking), the user agent must run the *time marches on* steps. If the [current playback position](#) changes while the steps are running, then the user agent must wait for the steps to complete, and then must immediately rerun the steps. (These steps are thus run as often as possible or needed — if one iteration takes a long time, this can cause certain [cues](#) to be skipped over as the user agent rushes ahead to "catch up".)

The *time marches on* steps are as follows:

1. Let *current cues* be a list of [cues](#), initialized to contain all the [cues](#) of all the [hidden](#) or [showing text tracks](#) of the [media element](#) (not the disabled ones) whose [start times](#) are less than or equal to the [current playback position](#) and whose [end times](#) are greater than the [current playback position](#).
2. Let *other cues* be a list of [cues](#), initialized to contain all the [cues](#) of [hidden](#) and [showing text tracks](#) of the [media element](#) that are not present in *current cues*.
3. Let *last time* be the [current playback position](#) at the time this algorithm was last run for this [media element](#), if this is not the first time it has run.
4. If the [current playback position](#) has, since the last time this algorithm was run, only changed through its usual monotonic increase during normal playback, then let *missed cues* be the list of [cues](#) in *other cues* whose [start times](#) are greater than or equal to *last time* and whose [end times](#) are less than or equal to the [current playback position](#). Otherwise, let *missed cues* be an empty list.
5. Remove all the [cues](#) in *missed cues* that are also in the [media element's list of newly introduced cues](#), and then empty the element's [list of newly introduced cues](#).

6. If the time was reached through the usual monotonic increase of the [current playback position](#) during normal playback, and if the user agent has not fired a `timeupdate` event at the element in the past 15 to 250ms and is not still running event handlers for such an event, then the user agent must [queue a task](#) to [fire a simple event](#) named `timeupdate` at the element. (In the other cases, such as explicit seeks, relevant events get fired as part of the overall process of changing the [current playback position](#).)

NOTE:

The event thus is not to be fired faster than about 66Hz or slower than 4Hz (assuming the event handlers don't take longer than 250ms to run). User agents are encouraged to vary the frequency of the event based on the system load and the average cost of processing the event each time, so that the UI updates are not any more frequent than the user agent can comfortably handle while decoding the video.

7. If all of the [cues](#) in [current cues](#) have their [text track cue active flag](#) set, none of the [cues](#) in [other cues](#) have their [text track cue active flag](#) set, and [missed cues](#) is empty, then abort these steps.
8. If the time was reached through the usual monotonic increase of the [current playback position](#) during normal playback, and there are [cues](#) in [other cues](#) that have their [text track cue pause-on-exit flag](#) set and that either have their [text track cue active flag](#) set or are also in [missed cues](#), then [immediately pause](#) the [media element](#).

NOTE:

In the other cases, such as explicit seeks, playback is not paused by going past the end time of a [cue](#), even if that [cue](#) has its [text track cue pause-on-exit flag](#) set.

9. Let [events](#) be a list of [tasks](#), initially empty. Each [task](#) in this list will be associated with a [text track](#), a [text track cue](#), and a time, which are used to sort the list before the [tasks](#) are queued.

Let [affected tracks](#) be a list of [text tracks](#), initially empty.

When the steps below say to **prepare an event** named *event* for a [text track cue](#) *target* with a time *time*, the user agent must run these substeps:

1. Let *track* be the [text track](#) with which the [text track cue](#) *target* is associated.
 2. Create a [task](#) to [fire a simple event](#) named *event* at *target*.
 3. Add the newly created [task](#) to [events](#), associated with the time *time*, the [text track](#) *track*, and the [text track cue](#) *target*.
 4. Add *track* to [affected tracks](#).
10. For each [text track cue](#) in [missed cues](#), [prepare an event](#) named `enter` for the `TextTrackCue` object with the [text track cue start time](#).
 11. For each [text track cue](#) in [other cues](#) that either has its [text track cue active flag](#) set or is in [missed cues](#), [prepare an event](#) named `exit` for the `TextTrackCue` object with the later of the [text track cue end time](#) and the [text track cue start time](#).
 12. For each [text track cue](#) in [current cues](#) that does not have its [text track cue active flag](#) set, [prepare an event](#) named `enter` for the `TextTrackCue` object with the [text track cue start time](#).
 13. Sort the [tasks](#) in [events](#) in ascending time order ([tasks](#) with earlier times first).

Further sort [tasks](#) in [events](#) that have the same time by the relative [text track cue order](#) of the [text track cues](#) associated with these [tasks](#).

Finally, sort [tasks](#) in [events](#) that have the same time and same [text track cue order](#) by placing [tasks](#) that fire `enter` events before those that fire `exit` events.
 14. [Queue](#) each [task](#) in [events](#), in list order.
 15. Sort [affected tracks](#) in the same order as the [text tracks](#) appear in the [media element](#)'s [list of text tracks](#), and remove duplicates.

16. For each [text track](#) in [affected tracks](#), in the list order, [queue a task](#) to [fire a simple event](#) named [cuechange](#) at the [TextTrack](#) object, and, if the [text track](#) has a corresponding [<track>](#) element, to then [fire a simple event](#) named [cuechange](#) at the [<track>](#) element as well.
17. Set the [text track cue active flag](#) of all the [cues](#) in the [current cues](#), and unset the [text track cue active flag](#) of all the [cues](#) in the [other cues](#).
18. Run the [rules for updating the text track rendering](#) of each of the [text tracks](#) in [affected tracks](#) that are [showing](#), providing the [text track's text track language](#) as the fallback language if it is not the empty string. For example, for [text tracks](#) based on WebVTT, the [rules for updating the display of WebVTT text tracks](#). [\[WEBVTT\]](#)

For the purposes of the algorithm above, a [text track cue](#) is considered to be part of a [text track](#) only if it is listed in the [text track list of cues](#), not merely if it is associated with the [text track](#).

NOTE:

If the [media element's](#) [node document](#) stops being a [fully active](#) document, then the playback will stop until the document is active again.

When a [media element](#) is [removed from a Document](#), the user agent must run the following steps:

1. [Await a stable state](#), allowing the [task](#) that removed the [media element](#) from the [Document](#) to continue. The [synchronous section](#) consists of all the remaining steps of this algorithm. (Steps in the [synchronous section](#) are marked with )
2.  If the [media element](#) is [in a Document](#), abort these steps.
3.  Run the [internal pause steps](#) for the [media element](#).

§ 4.7.14.9. Seeking

This definition is non-normative. Implementation requirements are given below this definition.

[media](#) . seeking

Returns true if the user agent is currently seeking.

[media](#) . seekable

Returns a [TimeRanges](#) object that represents the ranges of the [media resource](#) to which it is possible for the user agent to seek.

[media](#) . fastSeek(*time*)

Seeks to near the given *time* as fast as possible, trading precision for speed. (To seek to a precise time, use the [currentTime](#) attribute.)

This does nothing if the media resource has not been loaded.

The [seeking](#) attribute must initially have the value false.

The [fastSeek\(\)](#) method must [seek](#) to the time given by the method's argument, with the *approximate-for-speed* flag set.

When the user agent is required to [seek](#) to a particular *new playback position* in the [media resource](#), optionally with the *approximate-for-speed* flag set, it means that the user agent must run the following steps. This algorithm interacts closely with the [event loop](#) mechanism; in particular, it has a [synchronous section](#) (which is triggered as part of the [event loop](#) algorithm). Steps in that section are marked with .

1. Set the [media element's](#) [show poster flag](#) to false.
2. If the [media element's](#) [readyState](#) is [HAVE_NOTHING](#), abort these steps.
3. If the element's [seeking](#) IDL attribute is true, then another instance of this algorithm is already running. Abort that other instance of the algorithm without waiting for the step that it is running to complete.
4. Set the [seeking](#) IDL attribute to true.

5. If the seek was in response to a DOM method call or setting of an IDL attribute, then continue the script. The remainder of these steps must be run [in parallel](#). With the exception of the steps marked with ⏸, they could be aborted at any time by another instance of this algorithm being invoked.
6. If the *new playback position* is later than the end of the [media resource](#), then let it be the end of the [media resource](#) instead.
7. If the *new playback position* is less than the [earliest possible position](#), let it be that position instead.
8. If the (possibly now changed) *new playback position* is not in one of the ranges given in the `seekable` attribute, then let it be the position in one of the ranges given in the `seekable` attribute that is the nearest to the *new playback position*. If two positions both satisfy that constraint (i.e., the *new playback position* is exactly in the middle between two ranges in the `seekable` attribute) then use the position that is closest to the [current playback position](#). If there are no ranges given in the `seekable` attribute then set the seeking IDL attribute to false and abort these steps.
9. If the *approximate-for-speed* flag is set, adjust the *new playback position* to a value that will allow for playback to resume promptly. If *new playback position* before this step is before [current playback position](#), then the adjusted *new playback position* must also be before the [current playback position](#). Similarly, if the *new playback position* before this step is after [current playback position](#), then the adjusted *new playback position* must also be after the [current playback position](#).

EXAMPLE 423

For example, the user agent could snap to a nearby key frame, so that it doesn't have to spend time decoding then discarding intermediate frames before resuming playback.

10. [Queue a task](#) to [fire a simple event](#) named seeking at the element.
11. Set the [current playback position](#) to the *new playback position*.

NOTE:

If the [media element](#) was [potentially playing](#) immediately before it started seeking, but seeking caused its `readyState` attribute to change to a value lower than HAVE_FUTURE_DATA, then a waiting event will be [fired](#) at the element.

NOTE:

This step sets the [current playback position](#), and thus can immediately trigger other conditions, such as the rules regarding when playback "reaches the end of the [media resource](#)" (part of the logic that handles looping), even before the user agent is actually able to render the media data for that position (as determined in the next step).

NOTE:

The `currentTime` attribute returns the [official playback position](#), not the [current playback position](#), and therefore gets updated before script execution, separate from this algorithm.

12. Wait until the user agent has established whether or not the [media data](#) for the *new playback position* is available, and, if it is, until it has decoded enough data to play back that position.
13. [Await a stable state](#). The [synchronous section](#) consists of all the remaining steps of this algorithm. (Steps in the [synchronous section](#) are marked with ⏸.)
14. ⏸ Set the seeking IDL attribute to false.
15. ⏸ Run the [time marches on](#) steps.
16. ⏸ [Queue a task](#) to [fire a simple event](#) named timeupdate at the element.
17. ⏸ [Queue a task](#) to [fire a simple event](#) named seeked at the element.



The `seekable` attribute must return a new static [normalized TimeRanges object](#) that represents the ranges of the [media re-](#)

[source](#), if any, that the user agent is able to seek to, at the time the attribute is evaluated.

NOTE:

If the user agent can seek to anywhere in the [media resource](#), e.g., because it is a simple movie file and the user agent and the server support HTTP Range requests, then the attribute would return an object with one range, whose start is the time of the first frame (the [earliest possible position](#), typically zero), and whose end is the same as the time of the first frame plus the `duration` attribute's value (which would equal the time of the last frame, and might be positive Infinity).

NOTE:

The range might be continuously changing, e.g., if the user agent is buffering a sliding window on an infinite stream. This is the behavior seen with DVRs viewing live TV, for instance.

User agents should adopt a very liberal and optimistic view of what is seekable. User agents should also buffer recent content where possible to enable seeking to be fast.

EXAMPLE 424

For instance, consider a large video file served on an HTTP server without support for HTTP Range requests. A browser *could* implement this by only buffering the current frame and data obtained for subsequent frames, never allow seeking, except for seeking to the very start by restarting the playback. However, this would be a poor implementation. A high quality implementation would buffer the last few minutes of content (or more, if sufficient storage space is available), allowing the user to jump back and rewatch something surprising without any latency, and would in addition allow arbitrary seeking by reloading the file from the start if necessary, which would be slower but still more convenient than having to literally restart the video and watch it all the way through just to get to an earlier unbuffered spot.

[Media resources](#) might be internally scripted or interactive. Thus, a [media element](#) could play in a non-linear fashion. If this happens, the user agent must act as if the algorithm for [seeking](#) was used whenever the [current playback position](#) changes in a discontinuous fashion (so that the relevant events fire).

§ 4.7.14.10. Media resources with multiple media tracks

A [media resource](#) can have multiple embedded audio and video tracks. For example, in addition to the primary video and audio tracks, a [media resource](#) could have foreign-language dubbed dialogs, director's commentaries, audio descriptions, alternative angles, or sign-language overlays.

This definition is non-normative. Implementation requirements are given below this definition.

`media` . `audioTracks`

Returns an `AudioTrackList` object representing the audio tracks available in the [media resource](#).

`media` . `videoTracks`

Returns a `VideoTrackList` object representing the video tracks available in the [media resource](#).

The `audioTracks` attribute of a [media element](#) must return a [live](#) `AudioTrackList` object representing the audio tracks available in the [media element](#)'s [media resource](#).

The `videoTracks` attribute of a [media element](#) must return a [live](#) `VideoTrackList` object representing the video tracks available in the [media element](#)'s [media resource](#).

NOTE:

There are only ever one `AudioTrackList` object and one `VideoTrackList` object per [media element](#), even if another [media resource](#) is loaded into the element: the objects are reused. (The `AudioTrack` and `VideoTrack` objects are not, though.)

EXAMPLE 425

In this example, a script defines a function that takes a URL to a video and a reference to an element where the video is to be placed. That function then tries to load the video, and, once it is loaded, checks to see if there is a sign-language track available. If there is, it also displays that track. Both tracks are just placed in the given container; it's assumed that styles have been applied to make this work in a pretty way!

```
<script>
  function loadVideo(url, container) {
    var video = document.createElement('video');
    video.src = url;
    video.autoplay = true;
    video.controls = true;
    container.appendChild(video);
    video.onloadedmetadata = function (event) {
      for (var i = 0; i < video.videoTracks.length; i += 1) {
        if (video.videoTracks[i].kind == 'sign') {
          var sign = document.createElement('video');
          sign.src = url + '#track=' + video.videoTracks[i].id;
          sign.autoplay = true;
          container.appendChild(sign);
          return;
        }
      }
    };
  }
</script>
```

§ 4.7.14.10.1. AudioTrackList AND VideoTrackList OBJECTS

The AudioTrackList and VideoTrackList interfaces are used by attributes defined in the previous section.

```
interface AudioTrackList : EventTarget {
  readonly attribute unsigned long length;
  getter AudioTrack (unsigned long index);
  AudioTrack? getTrackById(DOMString id);

  attribute EventHandler onchange;
  attribute EventHandler onaddtrack;
  attribute EventHandler onremovetrack;
};
```

```
interface AudioTrack {
  readonly attribute DOMString id;
  readonly attribute DOMString kind;
  readonly attribute DOMString label;
  readonly attribute DOMString language;
  attribute boolean enabled;
};
```

```
interface VideoTrackList : EventTarget {
  readonly attribute unsigned long length;
  getter VideoTrack (unsigned long index);
  VideoTrack? getTrackById(DOMString id);
  readonly attribute long selectedIndex;

  attribute EventHandler onchange;
  attribute EventHandler onaddtrack;
  attribute EventHandler onremovetrack;
};
```

```
interface VideoTrack {
  readonly attribute DOMString id;
  readonly attribute DOMString kind;
  readonly attribute DOMString label;
  readonly attribute DOMString language;
  attribute boolean selected;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

`media . audioTracks . length`

`media . videoTracks . length`

Returns the number of tracks in the list.

`audioTrack = media . audioTracks[ index ]`

`videoTrack = media . videoTracks[ index ]`

Returns the specified AudioTrack or VideoTrack object.

`audioTrack = media . audioTracks . getTrackById( id )`

`videoTrack = media . videoTracks . getTrackById( id )`

Returns the AudioTrack or VideoTrack object with the given identifier, or null if no track has that identifier.

`audioTrack . id`

`videoTrack . id`

Returns the ID of the given track. This is the ID that can be used with a fragment identifier if the format supports the *Media Fragments URI* syntax, and that can be used with the `getTrackById()` method. [\[MEDIA-FRAGS\]](#)

`audioTrack . kind`

`videoTrack . kind`

Returns the category the given track falls into. The [possible track categories](#) are given below.

`audioTrack . label`

`videoTrack . label`

Returns the label of the given track, if known, or the empty string otherwise.

`audioTrack . language`

`videoTrack . language`

Returns the language of the given track, if known, or the empty string otherwise.

`audioTrack . enabled [ = value ]`

Returns true if the given track is active, and false otherwise.

Can be set, to change whether the track is enabled or not. If multiple audio tracks are enabled simultaneously, they are mixed.

`media . videoTracks . selectedIndex`

Returns the index of the currently selected track, if any, or -1 otherwise.

`videoTrack . selected [ = value ]`

Returns true if the given track is active, and false otherwise.

Can be set, to change whether the track is selected or not. Either zero or one video track is selected; selecting a new track while a previous one is selected will unselect the previous one.

An `AudioTrackList` object represents a dynamic list of zero or more audio tracks, of which zero or more can be enabled at a time. Each audio track is represented by an `AudioTrack` object.

A [VideoTrackList](#) object represents a dynamic list of zero or more video tracks, of which zero or one can be selected at a time. Each video track is represented by a `VideoTrack` object.

Tracks in `AudioTrackList` and [VideoTrackList](#) objects must be consistently ordered. If the [media resource](#) is in a format

that defines an order, then that order must be used; otherwise, the order must be the relative order in which the tracks are declared in the [media resource](#). The order used is called the *natural order* of the list.

NOTE:

Each track in one of these objects thus has an index; the first has the index 0, and each subsequent track is numbered one higher than the previous one. If a [media resource](#) dynamically adds or removes audio or video tracks, then the indices of the tracks will change dynamically. If the [media resource](#) changes entirely, then all the previous tracks will be removed and replaced with new tracks.

The `AudioTrackList.length` and `VideoTrackList.length` attributes must return the number of tracks represented by their objects at the time of getting.

The [supported property indices](#) of `AudioTrackList` and `VideoTrackList` objects at any instant are the numbers from zero to the number of tracks represented by the respective object minus one, if any tracks are represented. If an `AudioTrackList` or `VideoTrackList` object represents no tracks, it has no [supported property indices](#).

To [determine the value of an indexed property](#) for a given index *index* in an `AudioTrackList` or `VideoTrackList` object *list*, the user agent must return the `AudioTrack` or `VideoTrack` object that represents the *index*th track in *list*.

The `AudioTrackList.getTrackById(id)` and `VideoTrackList.getTrackById(id)` methods must return the first `AudioTrack` or `VideoTrack` object (respectively) in the `AudioTrackList` or `VideoTrackList` object (respectively) whose identifier is equal to the value of the *id* argument (in the natural order of the list, as defined above). When no tracks match the given argument, the methods must return null.

The `AudioTrack` and `VideoTrack` objects represent specific tracks of a [media resource](#). Each track can have an identifier, category, label, and language. These aspects of a track are permanent for the lifetime of the track; even if a track is removed from a [media resource](#)'s `AudioTrackList` or `VideoTrackList` objects, those aspects do not change.

In addition, `AudioTrack` objects can each be enabled or disabled; this is the audio track's *enabled state*. When an `AudioTrack` is created, its *enabled state* must be set to false (disabled). The [resource fetch algorithm](#) can override this.

Similarly, a single `VideoTrack` object per `VideoTrackList` object can be selected, this is the video track's *selection state*. When a `VideoTrack` is created, its *selection state* must be set to false (not selected). The [resource fetch algorithm](#) can override this.

The `AudioTrack.id` and `VideoTrack.id` attributes must return the identifier of the track, if it has one, or the empty string otherwise. If the [media resource](#) is in a format that supports the *Media Fragments URI* fragment identifier syntax, the identifier returned for a particular track must be the same identifier that would enable the track if used as the name of a track in the track dimension of such a fragment identifier. [\[MEDIA-FRAGS\]](#) [\[INBANDTRACKS\]](#)

EXAMPLE 426

For example, in Ogg files, this would be the Name header field of the track. [\[OGGSKELETON\]](#)

The `AudioTrack.kind` and `VideoTrack.kind` attributes must return the category of the track, if it has one, or the empty string otherwise.

The category of a track is the string given in the first column of the table below that is the most appropriate for the track based on the definitions in the table's second and third columns, as determined by the metadata included in the track in the [media resource](#). The cell in the third column of a row says what the category given in the cell in the first column of that row applies to; a category is only appropriate for an audio track if it applies to audio tracks, and a category is only appropriate for video tracks if it applies to video tracks. Categories must only be returned for `AudioTrack` objects if they are appropriate for audio, and must only be returned for `VideoTrack` objects if they are appropriate for video.

Return values for `AudioTrack.kind` and `VideoTrack.kind`

Category	Definition	Applies to...
"alternative"	A possible alternative to the main track, e.g., a different take of a song (audio), or a different angle (video).	Audio and video.
"captions"	A version of the main video track with captions burnt in. (For legacy content; new content would use text tracks.)	Video only.
"descriptions"	An audio description of a video track.	Audio only.

Category	Definition	Applies to...
"main"	The primary audio or video track.	Audio and video.
"main-desc"	The primary audio track, mixed with audio descriptions.	Audio only.
"sign"	A sign-language interpretation of an audio track.	Video only.
"subtitles"	A version of the main video track with subtitles burnt in. (For legacy content; new content would use text tracks.)	Video only.
"translation"	A translated version of the main audio track.	Audio only.
"commentary"	Commentary on the primary audio or video track, e.g., a director's commentary.	Audio and video.
"" (empty string)	No explicit kind, or the kind given by the track's metadata is not recognized by the user agent.	Audio and video.

The **AudioTrack.label** and **VideoTrack.label** attributes must return the label of the track, if it has one, or the empty string otherwise. [\[INBANDTRACKS\]](#)

The **AudioTrack.language** and **VideoTrack.language** attributes must return the BCP 47 language tag of the language of the track, if it has one, or the empty string otherwise. If the user agent is not able to express that language as a BCP 47 language tag (for example because the language information in the [media resource](#)'s format is a free-form string without a defined interpretation), then the method must return the empty string, as if the track had no language.

Source attribute values for id, kind, label and language of multitrack audio and video tracks as described for the relevant [media resource](#) format. [\[INBANDTRACKS\]](#)

The **AudioTrack.enabled** attribute, on getting, must return true if the track is currently enabled, and false otherwise. On setting, it must enable the track if the new value is true, and disable it otherwise. (If the track is no longer in an **AudioTrackList** object, then the track being enabled or disabled has no effect beyond changing the value of the attribute on the **AudioTrack** object.)

Whenever an audio track in an **AudioTrackList** that was disabled is enabled, and whenever one that was enabled is disabled, the user agent must [queue a task](#) to [fire a simple event](#) named `change` at the **AudioTrackList** object.

An audio track that has no data for a particular position on the [media timeline](#), or that does not exist at that position, must be interpreted as being silent at that point on the timeline.

The **VideoTrackList.selectedIndex** attribute must return the index of the currently selected track, if any. If the **VideoTrackList** object does not currently represent any tracks, or if none of the tracks are selected, it must instead return -1.

The **VideoTrack.selected** attribute, on getting, must return true if the track is currently selected, and false otherwise. On setting, it must select the track if the new value is true, and unselect it otherwise. If the track is in a **VideoTrackList**, then all the other **VideoTrack** objects in that list must be unselected. (If the track is no longer in a [VideoTrackList](#) object, then the track being selected or unselected has no effect beyond changing the value of the attribute on the **VideoTrack** object.)

Whenever a track in a **VideoTrackList** that was previously not selected is selected, and whenever the selected track in a **VideoTrackList** is unselected without a new track being selected in its stead, the user agent must [queue a task](#) to [fire a simple event](#) named `change` at the [VideoTrackList](#) object. This [task](#) must be [queued](#) before the [task](#) that fires the `resize` event, if any.

A video track that has no data for a particular position on the [media timeline](#) must be interpreted as being fully transparent black at that point on the timeline, with the same dimensions as the last frame before that position, or, if the position is before all the data for that track, the same dimensions as the first frame for that track. A track that does not exist at all at the current position must be treated as if it existed but had no data.

EXAMPLE 427

For instance, if a video has a track that is only introduced after one hour of playback, and the user selects that track then goes back to the start, then the user agent will act as if that track started at the start of the [media resource](#) but was simply transparent until one hour in.



The following are the [event handlers](#) (and their corresponding [event handler event types](#)) that must be supported, as [event handler IDL attributes](#), by all objects implementing the `AudioTrackList` and `VideoTrackList` interfaces:

Event handler	Event handler event type
<code>onchange</code>	<code>change</code>
<code>onaddtrack</code>	<code>addtrack</code>
<code>onremovetrack</code>	<code>removetrack</code>

§ 4.7.14.10.2. SELECTING SPECIFIC AUDIO AND VIDEO TRACKS DECLARATIVELY

The `audioTracks` and `videoTracks` attributes allow scripts to select which track should play, but it is also possible to select specific tracks declaratively, by specifying particular tracks in the fragment identifier of the [URL](#) of the [media resource](#). The format of the fragment identifier depends on the [MIME type](#) of the [media resource](#). [\[RFC2046\]](#) [\[URL\]](#)

EXAMPLE 428

In this example, a video that uses a format that supports the *Media Fragments URI* fragment identifier syntax is embedded in such a way that the alternative angles labeled "Alternative" are enabled instead of the default video track.

[\[MEDIA-FRAGS\]](#)

```
<video src="myvideo#track=Alternative"></video>
```

§ 4.7.14.11. Timed text tracks

§ 4.7.14.11.1. TEXT TRACK MODEL

A [media element](#) can have a group of associated **text tracks**, known as the [media element](#)'s **list of text tracks**. The [text tracks](#) are sorted as follows:

1. The [text tracks](#) corresponding to `<track>` element children of the [media element](#), in [tree order](#).
2. Any [text tracks](#) added using the `addTextTrack()` method, in the order they were added, oldest first.
3. Any [media-resource-specific text tracks](#) ([text tracks](#) corresponding to data in the [media resource](#)), in the order defined by the [media resource](#)'s format specification.

A [text track](#) consists of:

The kind of text track

This decides how the track is handled by the user agent. The kind is represented by a string. The possible strings are:

- **subtitles**
- **captions**
- **descriptions**
- **chapters**
- **metadata**

The [kind of track](#) can change dynamically, in the case of a [text track](#) corresponding to a `<track>` element.

A label

This is a human-readable string intended to identify the track for the user.

The [label of a track](#) can change dynamically, in the case of a [text track](#) corresponding to a `<track>` element.

When a [text track label](#) is the empty string, the user agent should automatically generate an appropriate label from the text track's other properties (e.g., the kind of text track and the text track's language) for use in its user interface. This automatically-generated label is not exposed in the API.

An in-band metadata track dispatch type

This is a string extracted from the [media resource](#) specifically for in-band metadata tracks to enable such tracks to be

dispatched to different scripts in the document.

EXAMPLE 429

For example, a traditional TV station broadcast streamed on the Web and augmented with Web-specific interactive features could include text tracks with metadata for ad targeting, trivia game data during game shows, player states during sports games, recipe information during food programs, and so forth. As each program starts and ends, new tracks might be added or removed from the stream, and as each one is added, the user agent could bind them to dedicated script modules using the value of this attribute.

Other than for in-band metadata text tracks, the [in-band metadata track dispatch type](#) is the empty string. How this value is populated for different media formats is described in [steps to expose a media-resource-specific text track](#).

A language

This is a string (a BCP 47 language tag) representing the language of the text track's cues. [\[BCP47\]](#)

The [language of a text track](#) can change dynamically, in the case of a [text track](#) corresponding to a `<track>` element.

A readiness state

One of the following:

Not loaded

Indicates that the text track's cues have not been obtained.

Loading

Indicates that the text track is loading and there have been no fatal errors encountered so far. Further cues might still be added to the track by the parser.

Loaded

Indicates that the text track has been loaded with no fatal errors.

Failed to load

Indicates that the text track was enabled, but when the user agent attempted to obtain it, this failed in some way (e.g., [URL](#) could not be [resolved](#), network error, unknown text track format). Some or all of the cues are likely missing and will not be obtained.

The [readiness state](#) of a [text track](#) changes dynamically as the track is obtained.

A mode

One of the following:

Disabled

Indicates that the text track is not active. Other than for the purposes of exposing the track in the DOM, the user agent is ignoring the text track. No cues are active, no events are fired, and the user agent will not attempt to obtain the track's cues.

Hidden

Indicates that the text track is active, but that the user agent is not actively displaying the cues. If no attempt has yet been made to obtain the track's cues, the user agent will perform such an attempt momentarily. The user agent is maintaining a list of which cues are active, and events are being fired accordingly.

Showing

Indicates that the text track is active. If no attempt has yet been made to obtain the track's cues, the user agent will perform such an attempt momentarily. The user agent is maintaining a list of which cues are active, and events are being fired accordingly. In addition, for text tracks whose [kind](#) is subtitles or captions, the cues are being overlaid on the video as appropriate; for text tracks whose [kind](#) is descriptions, the user agent is making the cues available to the user in a non-visual fashion; and for text tracks whose [kind](#) is chapters, the user agent is making available to the user a mechanism by which the user can navigate to any point in the [media resource](#) by selecting a cue.

A list of zero or more cues

A list of [text track cues](#), along with **rules for updating the text track rendering**. For example, for [WebVTT](#), the [rules for updating the display of WebVTT text tracks](#). [\[WEBVTT\]](#)

The [list of cues of a text track](#) can change dynamically, either because the [text track](#) has [not yet been loaded](#) or is still [loading](#), or due to DOM manipulation.

Each [text track](#) has a corresponding [TextTrack](#) object.



Each [media element](#) has a **list of pending text tracks**, which must initially be empty, a **blocked-on-parser** flag, which must initially be false, and a **did-perform-automatic-track-selection** flag, which must also initially be false.

When the user agent is required to **populate the list of pending text tracks** of a [media element](#), the user agent must add to the element's [list of pending text tracks](#) each [text track](#) in the element's [list of text tracks](#) whose [text track mode](#) is not disabled and whose [text track readiness state](#) is [loading](#).

Whenever a `<track>` element's parent node changes, the user agent must remove the corresponding [text track](#) from any [list of pending text tracks](#) that it is in.

Whenever a [text track](#)'s [text track readiness state](#) changes to either [loaded](#) or [failed to load](#), the user agent must remove it from any [list of pending text tracks](#) that it is in.

When a [media element](#) is created by an [HTML parser](#) or [XML parser](#), the user agent must set the element's [blocked-on-parser](#) flag to true. When a [media element](#) is popped off the [stack of open elements](#) of an [HTML parser](#) or [XML parser](#), the user agent must [honor user preferences for automatic text track selection](#), [populate the list of pending text tracks](#), and set the element's [blocked-on-parser](#) flag to false.

The [text tracks](#) of a [media element](#) are **ready** when both the element's [list of pending text tracks](#) is empty and the element's [blocked-on-parser](#) flag is false.

Each [media element](#) has a **pending text track change notification flag**, which must initially be unset.

Whenever a [text track](#) that is in a [media element](#)'s [list of text tracks](#) has its [text track mode](#) change value, the user agent must run the following steps for the [media element](#):

1. If the [media element](#)'s [pending text track change notification flag](#) is set, abort these steps.
2. Set the [media element](#)'s [pending text track change notification flag](#).
3. [Queue a task](#) that runs the following substeps:
 1. Unset the [media element](#)'s [pending text track change notification flag](#).
 2. [Fire a simple event](#) named `change` at the [media element](#)'s `textTracks` attribute's [TextTrackList](#) object.
4. If the [media element](#)'s [show poster flag](#) is not set, run the *time marches on* steps.

The [task source](#) for the [tasks](#) listed in this section is the [DOM manipulation task source](#).



A **text track cue** is the unit of time-sensitive data in a [text track](#), corresponding for instance for subtitles and captions to the text that appears at a particular time and disappears at another time.

Each [text track cue](#) consists of:

An identifier

An arbitrary string.

A start time

The time, in seconds and fractions of a second, that describes the beginning of the range of the [media data](#) to which the cue applies.

An end time

The time, in seconds and fractions of a second, that describes the end of the range of the [media data](#) to which the cue

applies.

A pause-on-exit flag

A boolean indicating whether playback of the [media resource](#) is to pause when the end of the range to which the cue applies is reached.

Some additional format-specific data

Additional fields, as needed for the format. For example, WebVTT has a [text track cue writing direction](#) and so forth. [\[WEBVTT\]](#)

Rules for extracting the chapter title

An algorithm which, when applied to the cue, returns a string that can be used in user interfaces that use the cue as a chapter title.

NOTE:

The [text track cue start time](#) and [text track cue end time](#) can be negative. (The [current playback position](#) can never be negative, though, so cues entirely before time zero cannot be active.)

Each [text track cue](#) has a corresponding `TextTrackCue` object (or more specifically, an object that inherits from `TextTrackCue` — for example, WebVTT cues use the `VTTcue` interface). A [text track cue](#)'s in-memory representation can be dynamically changed through this `TextTrackCue` API. [\[WEBVTT\]](#)

A [text track cue](#) is associated with [rules for updating the text track rendering](#), as defined by the specification for the specific kind of [text track cue](#). These rules are used specifically when the object representing the cue is added to a [TextTrack](#) object using the `addCue()` method.

In addition, each [text track cue](#) has two pieces of dynamic information:

The active flag

This flag must be initially unset. The flag is used to ensure events are fired appropriately when the cue becomes active or inactive, and to make sure the right cues are rendered.

The user agent must immediately unset this flag whenever the [text track cue](#) is removed from its [text track's text track list of cues](#); whenever the [text track](#) itself is removed from its [media element's list of text tracks](#) or has its [text track mode](#) changed to disabled; and whenever the [media element's](#) `readyState` is changed back to `HAVE_NOTHING`. When the flag is unset in this way for one or more cues in [text tracks](#) that were [showing](#) prior to the relevant incident, the user agent must, after having unset the flag for all the affected cues, apply the [rules for updating the text track rendering](#) of those [text tracks](#). For example, for [text tracks](#) based on [WebVTT](#), the [rules for updating the display of WebVTT text tracks](#). [\[WEBVTT\]](#)

The display state

This is used as part of the rendering model, to keep cues in a consistent position. It must initially be empty. Whenever the [text track cue active flag](#) is unset, the user agent must empty the [text track cue display state](#).

The [text track cues](#) of a [media element's text tracks](#) are ordered relative to each other in the [text track cue order](#), which is determined as follows: first group the [cues](#) by their [text track](#), with the groups being sorted in the same order as their [text tracks](#) appear in the [media element's list of text tracks](#); then, within each group, [cues](#) must be sorted by their [start time](#), earliest first; then, any [cues](#) with the same [start time](#) must be sorted by their [end time](#), latest first; and finally, any [cues](#) with identical [end times](#) must be sorted in the order they were last added to their respective [text track list of cues](#), oldest first (so e.g., for cues from a WebVTT file, that would initially be the order in which the cues were listed in the file). [\[WEBVTT\]](#)

§ 4.7.14.11.2. SOURCING IN-BAND TEXT TRACKS

A **media-resource-specific text track** is a [text track](#) that corresponds to data found in the [media resource](#).

Rules for processing and rendering such data are defined by the relevant specifications, e.g., the specification of the video format if the [media resource](#) is a video. Details for some legacy formats can be found in the *Sourcing In-band Media Resource Tracks from Media Containers into HTML* specification. [\[INBANDTRACKS\]](#)

When a [media resource](#) contains data that the user agent recognizes and supports as being equivalent to a [text track](#), the user agent [runs](#) the **steps to expose a media-resource-specific text track** with the relevant data, as follows.

1. Associate the relevant data with a new [text track](#) and its corresponding new [TextTrack](#) object. The [text track](#) is a [media-resource-specific text track](#).
2. Set the new [text track](#)'s [kind](#), [label](#), and [language](#) based on the semantics of the relevant data, as defined for the relevant format [\[INBANDTRACKS\]](#). If there is no label in that data, then the [label](#) must be set to the empty string.
3. Associate the [text track list of cues](#) with the [rules for updating the text track rendering](#) appropriate for the format in question.
4. If the new [text track](#)'s [kind](#) is metadata, then set the [text track in-band metadata track dispatch type](#) as follows, based on the type of the [media resource](#):

↪ If the [media resource](#) is an Ogg file

The [text track in-band metadata track dispatch type](#) must be set to the value of the Role header field. [\[OGGSKELETON\]](#)

↪ If the [media resource](#) is a WebM file

The [text track in-band metadata track dispatch type](#) must be set to the value of the CodecID element. [\[WEBM\]](#)

↪ If the [media resource](#) is an MPEG-2 file

Let *stream type* be the value of the "stream_type" field describing the text track's type in the file's program map section, interpreted as an 8-bit unsigned integer. Let *length* be the value of the "ES_info_length" field for the track in the same part of the program map section, interpreted as an integer as defined by the MPEG-2 specification. Let *descriptor bytes* be the *length* bytes following the "ES_info_length" field. The [text track in-band metadata track dispatch type](#) must be set to the concatenation of the *stream type* byte and the zero or more *descriptor bytes* bytes, expressed in hexadecimal using [uppercase ASCII hex digits](#). [\[MPEG2TS\]](#)

↪ If the [media resource](#) is an MPEG-4 file

Let the first *stsd* box of the first *stbl* box of the first *minf* box of the first *mdia* box of the [text track](#)'s *trak* box in the first *moov* box of the file be the *stsd box*, if any.

If the file has no *stsd box*, or if the *stsd box* has neither a *mett* box nor a *metx* box, then the [text track in-band metadata track dispatch type](#) must be set to the empty string.

Otherwise, if the *stsd box* has a *mett* box then the [text track in-band metadata track dispatch type](#) must be set to the concatenation of the string "mett", a U+0020 SPACE character, and the value of the first *mime_format* field of the first *mett* box of the *stsd box*, or the empty string if that field is absent in that box.

Otherwise, if the *stsd box* has no *mett* box but has a *metx* box then the [text track in-band metadata track dispatch type](#) must be set to the concatenation of the string "metx", a U+0020 SPACE character, and the value of the first *namespace* field of the first *metx* box of the *stsd box*, or the empty string if that field is absent in that box.

[\[MPEG4\]](#)

↪ If the [media resource](#) is a DASH media resource

The [text track in-band metadata track dispatch type](#) must be set to the concatenation of the "AdaptationSet" element attributes and all child Role descriptors. [\[MPEGDASH\]](#)

5. Populate the new [text track](#)'s [list of cues](#) with the cues parsed so far, following the [guidelines for exposing cues](#), and begin updating it dynamically as necessary.
6. Set the new [text track](#)'s [readiness state](#) to [loaded](#).
7. Set the new [text track](#)'s mode to the mode consistent with the user's preferences and the requirements of the relevant specification for the data.

NOTE:

For instance, if there are no other active subtitles, and this is a forced subtitle track (a subtitle track giving subtitles in the audio track's primary language, but only for audio that is actually in another language), then those subtitles might be activated here.

8. Add the new [text track](#) to the [media element](#)'s [list of text tracks](#).
9. Fire a [trusted](#) event with the name `addtrack`, that does not bubble and is not cancelable, and that uses the [TrackEvent](#) interface, with the `track` attribute initialized to the [text track](#)'s [TextTrack](#) object, at the [media element](#)'s `textTracks` attribute's [TextTrackList](#) object.

§ 4.7.14.11.3. SOURCING OUT-OF-BAND TEXT TRACKS

When a [<track>](#) element is created, it must be associated with a new [text track](#) (with its value set as defined below) and its corresponding new [TextTrack](#) object.

The [text track kind](#) is determined from the state of the element's `kind` attribute according to the following table; for a state given in a cell of the first column, the [kind](#) is the string given in the second column:

State	String
Subtitles	subtitles
Captions	captions
Descriptions	descriptions
Chapters	chapters
Metadata	metadata

The [text track label](#) is the element's [track label](#).

The [text track language](#) is the element's [track language](#), if any, or the empty string otherwise.

As the `kind`, `label`, and `srcLang` attributes are set, changed, or removed, the [text track](#) must update accordingly, as per the definitions above.

NOTE:

Changes to the [track URL](#) are handled in the algorithm below.

The [text track readiness state](#) is initially [not loaded](#), and the [text track mode](#) is initially disabled.

The [text track list of cues](#) is initially empty. It is dynamically modified when the referenced file is parsed. Associated with the list are the [rules for updating the text track rendering](#) appropriate for the format in question; for [WebVTT](#), this is the [rules for updating the display of WebVTT text tracks](#). [\[WEBVTT\]](#)

When a [<track>](#) element's parent element changes and the new parent is a [media element](#), then the user agent must add the [<track>](#) element's corresponding [text track](#) to the [media element](#)'s [list of text tracks](#), and then [queue a task to fire a trusted](#) event with the name `addtrack`, that does not bubble and is not cancelable, and that uses the [TrackEvent](#) interface, with the `track` attribute initialized to the [text track](#)'s [TextTrack](#) object, at the [media element](#)'s `textTracks` attribute's [TextTrackList](#) object.

When a [<track>](#) element's parent element changes and the old parent was a [media element](#), then the user agent must remove the [<track>](#) element's corresponding [text track](#) from the [media element](#)'s [list of text tracks](#), and then [queue a task to fire a trusted](#) event with the name `removetrack`, that does not bubble and is not cancelable, and that uses the [TrackEvent](#) interface, with the `track` attribute initialized to the [text track](#)'s [TextTrack](#) object, at the [media element](#)'s `textTracks` attribute's [TextTrackList](#) object.



When a [text track](#) corresponding to a [<track>](#) element is added to a [media element](#)'s [list of text tracks](#), the user agent must [queue a task](#) to run the following steps for the [media element](#):

1. If the element's [blocked-on-parser](#) flag is true, abort these steps.
2. If the element's [did-perform-automatic-track-selection](#) flag is true, abort these steps.
3. [Honor user preferences for automatic text track selection](#) for this element.

When the user agent is required to **honor user preferences for automatic text track selection** for a [media element](#), the user agent must run the following steps:

1. [Perform automatic text track selection](#) for subtitles and captions.
2. [Perform automatic text track selection](#) for descriptions.
3. [Perform automatic text track selection](#) for chapters.
4. If there are any [text tracks](#) in the [media element's list of text tracks](#) whose [text track kind](#) is metadata that correspond to track elements with a default attribute set whose [text track mode](#) is set to disabled, then set the [text track mode](#) of all such tracks to [hidden](#).
5. Set the element's [did-perform-automatic-track-selection](#) flag to true.

When the steps above say to **perform automatic text track selection** for one or more [text track kinds](#), it means to run the following steps:

1. Let *candidates* be a list consisting of the [text tracks](#) in the [media element's list of text tracks](#) whose [text track kind](#) is one of the kinds that were passed to the algorithm, if any, in the order given in the [list of text tracks](#).
2. If *candidates* is empty, then abort these steps.
3. If any of the [text tracks](#) in *candidates* have a [text track mode](#) set to [showing](#), abort these steps.
4. If the user has expressed an interest in having a track from *candidates* enabled based on its [text track kind](#), [text track language](#), and [text track label](#), then set its [text track mode](#) to [showing](#).

NOTE:

For example, the user could have set a browser preference to the effect of "I want French captions whenever possible", or "If there is a subtitle track with "Commentary" in the title, enable it", or "If there are audio description tracks available, enable one, ideally in Swiss German, but failing that in Standard Swiss German or Standard German".

Otherwise, if there are any [text tracks](#) in *candidates* that correspond to [<track>](#) elements with a default attribute set whose [text track mode](#) is set to disabled, then set the [text track mode](#) of the first such track to [showing](#).

When a [text track](#) corresponding to a [<track>](#) element experiences any of the following circumstances, the user agent must [start the track processing model](#) for that [text track](#) and its [<track>](#) element:

- The [<track>](#) element is created.
- The [text track](#) has its [text track mode](#) changed.
- The [<track>](#) element's parent element changes and the new parent is a [media element](#).

When a user agent is to **start the track processing model** for a [text track](#) and its [<track>](#) element, it must run the following algorithm. This algorithm interacts closely with the [event loop](#) mechanism; in particular, it has a [synchronous section](#) (which is triggered as part of the [event loop](#) algorithm). The steps in that section are marked with ⏸.

1. If another occurrence of this algorithm is already running for this [text track](#) and its [<track>](#) element, abort these steps, letting that other algorithm take care of this element.
2. If the [text track's text track mode](#) is not set to one of [hidden](#) or [showing](#), abort these steps.
3. If the [text track's <track>](#) element does not have a [media element](#) as a parent, abort these steps.
4. Run the remainder of these steps [in parallel](#), allowing whatever caused these steps to run to continue.
5. *Top: Await a stable state.* The [synchronous section](#) consists of the following steps. (The steps in the [synchronous section](#) are marked with ⏸.)
6. ⏸ Set the [text track readiness state](#) to [loading](#).
7. ⏸ Let *URL* be the [track URL](#) of the [<track>](#) element.

8.  If the `<track>` element's parent is a [media element](#) then let `corsAttributeState` be the state of the parent [media element](#)'s crossorigin content attribute. Otherwise, let `corsAttributeState` be [No CORS](#).
9. End the [synchronous section](#), continuing the remaining steps [in parallel](#).
10. If `URL` is not the empty string, run these substeps:
 1. Let `request` be the result of [creating a potential-CORS request](#) given `URL`, `corsAttributeState`, and with the *same-origin fallback flag* set.
 2. Set `request`'s `client` to the `<track>` element's [node document](#)'s Window object's [environment settings object](#) and `type` to "track".
 3. [Fetch](#) `request`.

The [tasks queued](#) by the [fetching algorithm](#) on the [networking task source](#) to process the data as it is being fetched must determine the type of the resource. If the type of the resource is not a supported text track format, the load will fail, as described below. Otherwise, the resource's data must be passed to the appropriate parser (e.g., the [WebVTT parser](#)) as it is received, with the [text track list of cues](#) being used for that parser's output. [\[WEBVTT\]](#)

NOTE:

The appropriate parser will incrementally update the [text track list of cues](#) during these [networking task source](#) tasks, as each such task is run with whatever data has been received from the network).

This specification does not currently say whether or how to check the MIME types of text tracks, or whether or how to perform file type sniffing using the actual file data. Implementors differ in their intentions on this matter and it is therefore unclear what the right solution is. In the absence of any requirement here, the HTTP specification's strict requirement to follow the Content-Type header prevails ("Content-Type specifies the media type of the underlying data." ... "If and only if the media type is not given by a Content-Type field, the recipient MAY attempt to guess the media type via inspection of its content and/or the name extension(s) of the URI used to identify the resource.").

If the [fetching algorithm](#) fails for any reason (network error, the server returns an error code, a cross-origin check fails, etc), or if `URL` is the empty string, then [queue a task](#) to first change the [text track readiness state](#) to [failed to load](#) and then [fire a simple event](#) named `error` at the `<track>` element. This [task](#) must use the [DOM manipulation task source](#).

If the [fetching algorithm](#) does not fail, but the type of the resource is not a supported text track format, or the file was not successfully processed (e.g., the format in question is an XML format and the file contained a well-formedness error that the XML specification requires be detected and reported to the application), then the [task](#) that is [queued](#) by the [networking task source](#) in which the aforementioned problem is found must change the [text track readiness state](#) to [failed to load](#) and [fire a simple event](#) named `error` at the `<track>` element.

If the [fetching algorithm](#) does not fail, and the file was successfully processed, then the final [task](#) that is [queued](#) by the [networking task source](#), after it has finished parsing the data, must change the [text track readiness state](#) to [loaded](#), and [fire a simple event](#) named `load` at the `<track>` element.

If, while fetching is ongoing, either:

- the [track URL](#) changes so that it is no longer equal to `URL`, while the [text track mode](#) is set to [hidden](#) or [showing](#); or
- the [text track mode](#) changes to [hidden](#) or [showing](#), while the [track URL](#) is not equal to `URL`

...then the user agent must abort [fetching](#), discarding any pending [tasks](#) generated by that algorithm (and in particular, not adding any cues to the [text track list of cues](#) after the moment the URL changed), and then [queue a task](#) that first changes the [text track readiness state](#) to [failed to load](#) and then [fires a simple event](#) named `error` at the `<track>` element. This [task](#) must use the [DOM manipulation task source](#).

11. Wait until the [text track readiness state](#) is no longer set to [loading](#).
12. Wait until the [track URL](#) is no longer equal to `URL`, at the same time as the [text track mode](#) is set to [hidden](#) or [show-](#)

ing.

13. Jump to the step labeled *top*.

Whenever a `<track>` element has its `src` attribute set, changed, or removed, the user agent must [immediately](#) empty the element's [text track](#)'s [text track list of cues](#). (This also causes the algorithm above to stop adding cues from the resource being obtained using the previously given URL, if any.)

§ 4.7.14.11.4. GUIDELINES FOR EXPOSING CUES IN VARIOUS FORMATS AS [TEXT TRACK CUES](#)

How a specific format's text track cues are to be interpreted for the purposes of processing by an HTML user agent is defined by that format [\[INBANDTRACKS\]](#). In the absence of such a specification, this section provides some constraints within which implementations can attempt to consistently expose such formats.

To support the [text track](#) model of HTML, each unit of timed data is converted to a [text track cue](#). Where the mapping of the format's features to the aspects of a [text track cue](#) as defined in this specification are not defined, implementations must ensure that the mapping is consistent with the definitions of the aspects of a [text track cue](#) as defined above, as well as with the following constraints:

The [text track cue identifier](#)

Should be set to the empty string if the format has no obvious analog to a per-cue identifier.

The [text track cue pause-on-exit flag](#)

Should be set to false.

For [media-resource-specific text tracks](#) of [kind](#) metadata, [text track cues](#) are exposed using the `DataCue` object unless there is a more appropriate `TextTrackCue` interface available. For example, if the [media-resource-specific text track](#) format is [WebVTT](#), then `VTTcue` is more appropriate.

§ 4.7.14.11.5. TEXT TRACK API

```
interface TextTrackList : EventTarget {
  readonly attribute unsigned long length;
  getter TextTrack (unsigned long index);
  TextTrack? getTrackById(DOMString id);

  attribute EventHandler onchange;
  attribute EventHandler onaddtrack;
  attribute EventHandler onremovetrack;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

`media . textTracks . length`

Returns the number of [text tracks](#) associated with the [media element](#) (e.g., from `<track>` elements). This is the number of [text tracks](#) in the [media element](#)'s [list of text tracks](#).

`media . textTracks[ n ]`

Returns the [TextTrack](#) object representing the *n*th [text track](#) in the [media element](#)'s [list of text tracks](#).

`textTrack = media . textTracks . getTrackById( id )`

Returns the [TextTrack](#) object with the given identifier, or null if no track has that identifier.

A [TextTrackList](#) object represents a dynamically updating list of [text tracks](#) in a given order.

The `textTracks` attribute of [media elements](#) must return a [TextTrackList](#) object representing the [TextTrack](#) objects of the [text tracks](#) in the [media element](#)'s [list of text tracks](#), in the same order as in the [list of text tracks](#).

The `length` attribute of a [TextTrackList](#) object must return the number of [text tracks](#) in the list represented by the [TextTrackList](#) object.

The [supported property indices](#) of a [TextTrackList](#) object at any instant are the numbers from zero to the number of [text tracks](#) in the list represented by the [TextTrackList](#) object minus one, if any. If there are no [text tracks](#) in the list, there are no [supported property indices](#).

To [determine the value of an indexed property](#) of a [TextTrackList](#) object for a given index *index*, the user agent must return the *index*th [text track](#) in the list represented by the [TextTrackList](#) object.

The [getTrackById\(id\)](#) method must return the first [TextTrack](#) in the [TextTrackList](#) object whose [id](#) IDL attribute would return a value equal to the value of the *id* argument. When no tracks match the given argument, the method must return null.



```
enum TextTrackMode { "disabled", "hidden", "showing" };

enum TextTrackKind { "subtitles", "captions", "descriptions", "chapters", "metadata" };

interface TextTrack : EventTarget {
  readonly attribute TextTrackKind kind;
  readonly attribute DOMString label;
  readonly attribute DOMString language;

  readonly attribute DOMString id;
  readonly attribute DOMString inBandMetadataTrackDispatchType;

  attribute TextTrackMode mode;

  readonly attribute TextTrackCueList? cues;
  readonly attribute TextTrackCueList? activeCues;

  void addCue(TextTrackCue cue);
  void removeCue(TextTrackCue cue);

  attribute EventHandler oncuechange;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

`textTrack = media . addTextTrack( kind [, label [, language ] ] )`

Creates and returns a new [TextTrack](#) object, which is also added to the [media element](#)'s [list of text tracks](#).

`textTrack . kind`

Returns the [text track kind](#) string.

`textTrack . label`

Returns the [text track label](#), if there is one, or the empty string otherwise (indicating that a custom label probably needs to be generated from the other attributes of the object if the object is exposed to the user).

`textTrack . language`

Returns the [text track language](#) string.

`textTrack . id`

Returns the ID of the given track.

For in-band tracks, this is the ID that can be used with a fragment identifier if the format supports the *Media Fragments URI* syntax, and that can be used with the [getTrackById\(\)](#) method. [\[MEDIA-FRAGS\]](#)

For [TextTrack](#) objects corresponding to [<track>](#) elements, this is the ID of the [<track>](#) element.

`textTrack . inBandMetadataTrackDispatchType`

Returns the [text track in-band metadata track dispatch type](#) string.

`textTrack . mode [ = value ]`

Returns the [text track mode](#), represented by a string from the following list:

"disabled"

The [text track disabled](#) mode.

"hidden"

The [text track hidden](#) mode.

"showing"

The [text track showing](#) mode.

Can be set, to change the mode.

`textTrack . cues`

Returns the [text track list of cues](#), as a `TextTrackCueList` object.

`textTrack . activeCues`

Returns the [text track cues](#) from the [text track list of cues](#) that are currently active (i.e., that start before the [current playback position](#) and end after it), as a `TextTrackCueList` object.

`textTrack . addCue( cue )`

Adds the given cue to `textTrack`'s [text track list of cues](#).

`textTrack . removeCue( cue )`

Removes the given cue from `textTrack`'s [text track list of cues](#).

The `addTextTrack( kind , label , language )` method of [media elements](#), when invoked, must run the following steps:

1. Create a new [TextTrack](#) object.
2. Create a new [text track](#) corresponding to the new object, and set its [text track kind](#) to `kind`, its [text track label](#) to `label`, its [text track language](#) to `language`, its [text track readiness state](#) to the [text track loaded](#) state, its [text track mode](#) to the [text track hidden](#) mode, and its [text track list of cues](#) to an empty list.

Initially, the [text track list of cues](#) is not associated with any [rules for updating the text track rendering](#). When a [text track cue](#) is added to it, the [text track list of cues](#) has its rules permanently set accordingly.

3. Add the new [text track](#) to the [media element](#)'s [list of text tracks](#).
4. Queue a task to fire a [trusted](#) event with the name `addtrack`, that does not bubble and is not cancelable, and that uses the [TrackEvent](#) interface, with the `track` attribute initialized to the new [text track](#)'s [TextTrack](#) object, at the [media element](#)'s `textTracks` attribute's `TextTrackList` object.
5. Return the new [TextTrack](#) object.



The `kind` attribute must return the [text track kind](#) of the [text track](#) that the [TextTrack](#) object represents.

The `label` attribute must return the [text track label](#) of the [text track](#) that the [TextTrack](#) object represents.

The `language` attribute must return the [text track language](#) of the [text track](#) that the [TextTrack](#) object represents.

The `id` attribute returns the track's identifier, if it has one, or the empty string otherwise. For tracks that correspond to `<track>` elements, the track's identifier is the value of the element's `id` attribute, if any. For in-band tracks, the track's identifier is specified by the [media resource](#). If the [media resource](#) is in a format that supports the *Media Fragments URI* fragment identifier syntax, the identifier returned for a particular track must be the same identifier that would enable the track if used as the name of a track in the track dimension of such a fragment identifier. [\[MEDIA-FRAGS\]](#)

The `inBandMetadataTrackDispatchType` attribute must return the [text track in-band metadata track dispatch type](#) of the [text track](#) that the [TextTrack](#) object represents.

The **mode** attribute, on getting, must return the string corresponding to the [text track mode](#) of the [text track](#) that the [TextTrack](#) object represents, as defined by the following list:

"disabled"

The [text track disabled](#) mode.

"hidden"

The [text track hidden](#) mode.

"showing"

The [text track showing](#) mode.

On setting, if the new value isn't equal to what the attribute would currently return, the new value must be processed as follows:

↪ **If the new value is "disabled"**

Set the [text track mode](#) of the [text track](#) that the [TextTrack](#) object represents to the [text track disabled](#) mode.

↪ **If the new value is "hidden"**

Set the [text track mode](#) of the [text track](#) that the [TextTrack](#) object represents to the [text track hidden](#) mode.

↪ **If the new value is "showing"**

Set the [text track mode](#) of the [text track](#) that the [TextTrack](#) object represents to the [text track showing](#) mode.

If the [text track mode](#) of the [text track](#) that the [TextTrack](#) object represents is not the [text track disabled](#) mode, then the **cues** attribute must return a [live](#) [TextTrackCueList](#) object that represents the subset of the [text track list of cues](#) of the [text track](#) that the [TextTrack](#) object represents whose [end times](#) occur at or after the [earliest possible position when the script started](#), in [text track cue order](#). Otherwise, it must return null. For each [TextTrack](#) object, when an object is returned, the same [TextTrackCueList](#) object must be returned each time.

The [earliest possible position when the script started](#) is whatever the [earliest possible position](#) was the last time the [event loop](#) reached step 1.

If the [text track mode](#) of the [text track](#) that the [TextTrack](#) object represents is not the [text track disabled](#) mode, then the **activeCues** attribute must return a [live](#) [TextTrackCueList](#) object that represents the subset of the [text track list of cues](#) of the [text track](#) that the [TextTrack](#) object represents whose [active flag was set when the script started](#), in [text track cue order](#). Otherwise, it must return null. For each [TextTrack](#) object, when an object is returned, the same [TextTrackCueList](#) object must be returned each time.

A [text track cue](#)'s [active flag was set when the script started](#) if its [text track cue active flag](#) was set the last time the [event loop](#) reached step 1.



The **addCue(cue)** method of [TextTrack](#) objects, when invoked, must run the following steps:

1. If the [text track list of cues](#) does not yet have any associated [rules for updating the text track rendering](#), then associate the [text track list of cues](#) with the [rules for updating the text track rendering](#) appropriate to *cue*.
2. If [text track list of cues](#)' associated [rules for updating the text track rendering](#) are not the same [rules for updating the text track rendering](#) as appropriate for *cue*, then throw an `InvalidStateError` exception and abort these steps.
3. If the given *cue* is in a [text track list of cues](#), then remove *cue* from that [text track list of cues](#).
4. Add *cue* to the method's [TextTrack](#) object's [text track](#)'s [text track list of cues](#).

The **removeCue(cue)** method of [TextTrack](#) objects, when invoked, must run the following steps:

1. If the given *cue* is not currently listed in the method's [TextTrack](#) object's [text track](#)'s [text track list of cues](#), then throw a `NotFoundError` exception and abort these steps.
2. Remove *cue* from the method's [TextTrack](#) object's [text track](#)'s [text track list of cues](#).

EXAMPLE 430

In this example, an `<audio>` element is used to play a specific sound-effect from a sound file containing many sound effects. A cue is used to pause the audio, so that it ends exactly at the end of the clip, even if the browser is busy running some script. If the page had relied on script to pause the audio, then the start of the next clip might be heard if the browser was not able to run the script at the exact time specified.

```
var sfx = new Audio('sfx.wav');
var sounds = sfx.addTextTrack('metadata');

// add sounds we care about
function addFX(start, end, name) {
  var cue = new VTTCue(start, end, '');
  cue.id = name;
  cue.pauseOnExit = true;
  sounds.addCue(cue);
}
addFX(12.783, 13.612, 'dog bark');
addFX(13.612, 15.091, 'kitten mew'))

function playSound(id) {
  sfx.currentTime = sounds.getCueById(id).startTime;
  sfx.play();
}

// play a bark as soon as we can
sfx.oncanplaythrough = function () {
  playSound('dog bark');
}
// meow when the user tries to leave
window.onbeforeunload = function () {
  playSound('kitten mew');
  return 'Are you sure you want to leave this awesome page?';
}
```



```
interface TextTrackCueList {
  readonly attribute unsigned long length;
  getter TextTrackCue (unsigned long index);
  TextTrackCue? getCueById(DOMString id);
};
```

This definition is non-normative. Implementation requirements are given below this definition.

`cuelist . length`

Returns the number of [cues](#) in the list.

`cuelist [index]`

Returns the [text track cue](#) with index `index` in the list. The cues are sorted in [text track cue order](#).

`cuelist . getCueById( id )`

Returns the first [text track cue](#) (in [text track cue order](#)) with [text track cue identifier](#) `id`. Returns null if none of the cues have the given identifier or if the argument is the empty string.

A `TextTrackCueList` object represents a dynamically updating list of [text track cues](#) in a given order.

The **`length`** attribute must return the number of [cues](#) in the list represented by the `TextTrackCueList` object.

The [supported property indices](#) of a `TextTrackCueList` object at any instant are the numbers from zero to the number of

[cues](#) in the list represented by the `TextTrackCueList` object minus one, if any. If there are no [cues](#) in the list, there are no [supported property indices](#).

To determine the value of an indexed property for a given index *index*, the user agent must return the *index*th [text track cue](#) in the list represented by the `TextTrackCueList` object.

The `getCueById( id )` method, when called with an argument other than the empty string, must return the first [text track cue](#) in the list represented by the `TextTrackCueList` object whose [text track cue identifier](#) is *id*, if any, or null otherwise. If the argument is the empty string, then the method must return null.



```
interface TextTrackCue : EventTarget {
  readonly attribute TextTrack? track;

  attribute DOMString id;
  attribute double startTime;
  attribute double endTime;
  attribute boolean pauseOnExit;

  attribute EventHandler onenter;
  attribute EventHandler onexit;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

`cue . track`

Returns the [TextTrack](#) object to which this [text track cue](#) belongs, if any, or null otherwise.

`cue . id [ = value ]`

Returns the [text track cue identifier](#). Can be set.

`cue . startTime [ = value ]`

Returns the [text track cue start time](#), in seconds. Can be set.

`cue . endTime [ = value ]`

Returns the [text track cue end time](#), in seconds. Can be set.

`cue . pauseOnExit [ = value ]`

Returns true if the [text track cue pause-on-exit flag](#) is set, false otherwise. Can be set.

The `track` attribute, on getting, must return the [TextTrack](#) object of the [text track](#) in whose [list of cues](#) the [text track cue](#) that the `TextTrackCue` object represents finds itself, if any; or null otherwise.

The `id` attribute, on getting, must return the [text track cue identifier](#) of the [text track cue](#) that the `TextTrackCue` object represents. On setting, the [text track cue identifier](#) must be set to the new value.

The `startTime` attribute, on getting, must return the [text track cue start time](#) of the [text track cue](#) that the `TextTrackCue` object represents, in seconds. On setting, the [text track cue start time](#) must be set to the new value, interpreted in seconds; then, if the `TextTrackCue` object's [text track cue](#) is in a [text track](#)'s [list of cues](#), and that [text track](#) is in a [media element](#)'s [list of text tracks](#), and the [media element](#)'s [show poster flag](#) is not set, then run the *time marches on* steps for that [media element](#).

The `endTime` attribute, on getting, must return the [text track cue end time](#) of the [text track cue](#) that the `TextTrackCue` object represents, in seconds. On setting, the [text track cue end time](#) must be set to the new value, interpreted in seconds; then, if the `TextTrackCue` object's [text track cue](#) is in a [text track](#)'s [list of cues](#), and that [text track](#) is in a [media element](#)'s [list of text tracks](#), and the [media element](#)'s [show poster flag](#) is not set, then run the *time marches on* steps for that [media element](#).

The `pauseOnExit` attribute, on getting, must return true if the [text track cue pause-on-exit flag](#) of the [text track cue](#) that the `TextTrackCue` object represents is set; or false otherwise. On setting, the [text track cue pause-on-exit flag](#) must be set if the new value is true, and must be unset otherwise.

§ 4.7.14.11.6. TEXT TRACKS EXPOSING IN-BAND METADATA

⚠Warning! The use of text tracks exposing in-band metadata is "at risk". If testing during the Candidate Recommendation phase does not identify at least two interoperable implementations in current shipping browsers of text tracks exposing in-band metadata this section will be removed from the HTML 5.1 Specification.

[Media resources](#) often contain one or more [media-resource-specific text tracks](#) containing data that browsers don't render, but want to expose to script to allow being dealt with.

If the browser is unable to identify a `TextTrackCue` interface that is more appropriate to expose the data in the cues of a [media-resource-specific text track](#), the `DataCue` object is used. [\[INBANDTRACKS\]](#)

```
[Constructor(double startTime, double endTime, ArrayBuffer data)]
interface DataCue : TextTrackCue {
  attribute ArrayBuffer data;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

```
cue = new DataCue( [ startTime, endTime, data ] )
```

Returns a new `DataCue` object, for use with the `addCue()` method. The `startTime` argument sets the [text track cue start time](#). The `endTime` argument sets the [text track cue end time](#). The `data` argument is copied as the [text track cue data](#).

```
cue . data [ = value ]
```

Returns the [text track cue data](#) in raw unparsed form. Can be set.

The **data** attribute, on getting, must return the raw text track cue data of the [text track cue](#) that the `TextTrackCue` object represents. On setting, the text track cue data must be set to the new value.

The user agent will use `DataCue` to expose only [text track cue](#) objects that belong to a [text track](#) that has a [text track kind](#) of [metadata](#).

NOTE:

`DataCue` has a constructor to allow script to create `DataCue` objects in cases where generic metadata needs to be managed for a [text track](#).

The [rules for updating the text track rendering](#) for a `DataCue` simply state that there is no rendering, even when the cues are in [showing](#) mode and the [text track kind](#) is one of subtitles or captions or descriptions or chapters.

§ 4.7.14.11.7. TEXT TRACKS DESCRIBING CHAPTERS

Chapters are segments of a [media resource](#) with a given title. Chapters can be nested, in the same way that sections in a document outline can have subsections.

Each [text track cue](#) in a [text track](#) being used for describing chapters has three key features: the [text track cue start time](#), giving the start time of the chapter, the [text track cue end time](#), giving the end time of the chapter, and the [text track rules for extracting the chapter title](#).

The **rules for constructing the chapter tree from a text track** are as follows. They produce a potentially nested list of chapters, each of which have a start time, end time, title, and a list of nested chapters. This algorithm discards cues that do not correctly nest within each other, or that are out of order.

1. Let *list* be a copy of the [list of cues](#) of the [text track](#) being processed.
2. Remove from *list* any [text track cue](#) whose [text track cue end time](#) is before its [text track cue start time](#).
3. Let *output* be an empty list of chapters, where a chapter is a record consisting of a start time, an end time, a title, and a (potentially empty) list of nested chapters. For the purpose of this algorithm, each chapter also has a parent chapter.

4. Let *current chapter* be a stand-in chapter whose start time is negative infinity, whose end time is positive infinity, and whose list of nested chapters is *output*. (This is just used to make the algorithm easier to describe.)
5. *Loop*: If *list* is empty, jump to the step labeled *end*.
6. Let *current cue* be the first cue in *list*, and then remove it from *list*.
7. If *current cue*'s [text track cue start time](#) is less than the start time of *current chapter*, then return to the step labeled *loop*.
8. While *current cue*'s [text track cue start time](#) is greater than or equal to *current chapter*'s end time, let *current chapter* be *current chapter*'s parent chapter.
9. If *current cue*'s [text track cue end time](#) is greater than the end time of *current chapter*, then return to the step labeled *loop*.
10. Create a new chapter *new chapter*, whose start time is *current cue*'s [text track cue start time](#), whose end time is *current cue*'s [text track cue end time](#), whose title is *current cue*'s [text track cue data](#) interpreted according to its rules for rendering the cue in isolation, and whose list of nested chapters is empty.
11. Append *new chapter* to *current chapter*'s list of nested chapters, and let *current chapter* be *new chapter*'s parent.
12. Let *current chapter* be *new chapter*.
13. Return to the step labeled *loop*.
14. *End*: Return *output*.

EXAMPLE 431

The following snippet of a [WebVTT file](#) shows how nested chapters can be marked up. The file describes three 50-minute chapters, "Astrophysics", "Computational Physics", and "General Relativity". The first has three subchapters, the second has four, and the third has two. [\[WEBVTT\]](#)

```
WEBVTT
00:00:00.000 --> 00:50:00.000
Astrophysics

00:00:00.000 --> 00:10:00.000
Introduction to Astrophysics

00:10:00.000 --> 00:45:00.000
The Solar System

00:00:00.000 --> 00:10:00.000
Coursework Description

00:50:00.000 --> 01:40:00.000
Computational Physics

00:50:00.000 --> 00:55:00.000
Introduction to Programming

00:55:00.000 --> 01:30:00.000
Data Structures

01:30:00.000 --> 01:35:00.000
Answers to Last Exam

01:35:00.000 --> 01:40:00.000
Coursework Description

01:40:00.000 --> 02:30:00.000
General Relativity

01:40:00.000 --> 02:00:00.000
Tensor Algebra

02:00:00.000 --> 02:30:00.000
The General Relativistic Field Equations
```

§ 4.7.14.11.8. EVENT HANDLERS FOR OBJECTS OF THE TEXT TRACK APIs

The following are the [event handlers](#) that (and their corresponding [event handler event types](#)) must be supported, as [event handler IDL attributes](#), by all objects implementing the `TextTrackList` interface:

Event handler	Event handler event type
onchange	change
onaddtrack	addtrack
onremovetrack	removetrack

The following are the [event handlers](#) that (and their corresponding [event handler event types](#)) must be supported, as [event handler IDL attributes](#), by all objects implementing the `TextTrack` interface:

Event handler	Event handler event type
oncuechange	cuechange

The following are the [event handlers](#) that (and their corresponding [event handler event types](#)) must be supported, as [event handler IDL attributes](#), by all objects implementing the TextTrackCue interface:

Event handler	Event handler event type
onenter	enter
onexit	exit

§ 4.7.14.11.9. BEST PRACTICES FOR METADATA TEXT TRACKS

This section is non-normative.

Text tracks can be used for storing data relating to the media data, for interactive or augmented views.

For example, a page showing a sports broadcast could include information about the current score. Suppose a robotics competition was being streamed live. The image could be overlaid with the scores, as follows:



In order to make the score display render correctly whenever the user seeks to an arbitrary point in the video, the metadata text track cues need to be as long as is appropriate for the score. For example, in the frame above, there would be maybe one cue that lasts the length of the match that gives the match number, one cue that lasts until the blue alliance's score changes, and one cue that lasts until the red alliance's score changes. If the video is just a stream of the live event, the time in the bottom right would presumably be automatically derived from the current video time, rather than based on a cue. However, if the video was just the highlights, then that might be given in cues also.

The following shows what fragments of this could look like in a WebVTT file:


```

WEBVTT
...

05:10:00.000 --> 05:12:15.000
matchtype:qual
matchnumber:37

...

05:11:02.251 --> 05:11:17.198
red:78

05:11:03.672 --> 05:11:54.198
blue:66

05:11:17.198 --> 05:11:25.912
red:80

05:11:25.912 --> 05:11:26.522
red:83

05:11:26.522 --> 05:11:26.982
red:86

05:11:26.982 --> 05:11:27.499
red:89

...

```

The key here is to notice that the information is given in cues that span the length of time to which the relevant event applies. If, instead, the scores were given as zero-length (or very brief, nearly zero-length) cues when the score changes, for example saying "red+2" at 05:11:17.198, "red+3" at 05:11:25.912, etc, problems arise: primarily, seeking is much harder to implement, as the script has to walk the entire list of cues to make sure that no notifications have been missed; but also, if the cues are short it's possible the script will never see that they are active unless it listens to them specifically.

When using cues in this manner, authors are encouraged to use the `cuechange` event to update the current annotations. (In particular, using the `timeupdate` event would be less appropriate as it would require doing work even when the cues haven't changed, and, more importantly, would introduce a higher latency between when the metadata cues become active and when the display is updated, since `timeupdate` events are rate-limited.)

§ 4.7.14.12. *User interface*

The **controls** attribute is a [boolean attribute](#). If present, it indicates that the author has not provided a scripted controller and would like the user agent to provide its own set of controls.

If the attribute is present, or if [scripting is disabled](#) for the [media element](#), then the user agent should **expose a user interface to the user**. This user interface should include features to begin playback, pause playback, seek to an arbitrary position in the content (if the content supports arbitrary seeking), change the volume, change the display of closed captions or embedded sign-language tracks, select different audio tracks or turn on audio descriptions, and show the media content in manners more suitable to the user (e.g., fullscreen video or in an independent resizable window). Other controls may also be made available.

A user agent may provide controls to affect playback of the media resource (e.g., play, pause, seeking, track selection, and volume controls), but such features should not interfere with the page's normal rendering. For example, such features could be exposed in the [media element](#)'s context menu, platform media keys, or a remote control. The user agent may implement this simply by [exposing a user interface to the user](#) as described above (as if the [controls](#) attribute was present).

If the user agent [exposes a user interface to the user](#) by displaying controls over the [media element](#), then the user agent should suppress any user interaction events while the user agent is interacting with this interface. (For example, if the user clicks on a video's playback control, `mousedown` events and so forth would not simultaneously be fired at elements on the page.)

Where possible (specifically, for starting, stopping, pausing, and unpausing playback, for seeking, for changing the rate of playback, for fast-forwarding or rewinding, for listing, enabling, and disabling text tracks, and for muting or changing the volume of the audio), user interface features exposed by the user agent must be implemented in terms of the DOM API described above, so that, e.g., all the same events fire.

For the purposes of listing chapters in the [media resource](#), only [text tracks](#) in the [media element](#)'s [list of text tracks](#) that are [showing](#) and whose [text track kind](#) is [chapters](#) should be used. Such tracks must be interpreted according to the [rules for constructing the chapter tree from a text track](#). When seeking in response to a user manipulating a chapter selection interface, user agents should not use the *approximate-for-speed* flag.

The **controls** IDL attribute must [reflect](#) the content attribute of the same name.



This definition is non-normative. Implementation requirements are given below this definition.

***media* . volume [= *value*]**

Returns the current playback volume, as a number in the range 0.0 to 1.0, where 0.0 is the quietest and 1.0 the loudest.

Can be set, to change the volume.

Throws an `IndexSizeError` exception if the new value is not in the range 0.0 .. 1.0.

***media* . muted [= *value*]**

Returns true if audio is muted, overriding the `volume` attribute, and false if the `volume` attribute is being honored.

Can be set, to change whether the audio is muted or not.

A [media element](#) has a **playback volume**, which is a fraction in the range 0.0 (silent) to 1.0 (loudest). Initially, the volume should be 1.0, but user agents may remember the last set value across sessions, on a per-site basis or otherwise, so the volume may start at other values.

The **volume** IDL attribute must return the [playback volume](#) of any audio portions of the [media element](#). On setting, if the new value is in the range 0.0 to 1.0 inclusive, the [media element](#)'s [playback volume](#) must be set to the new value. If the new value is outside the range 0.0 to 1.0 inclusive, then, on setting, an `IndexSizeError` exception must be thrown instead.

A [media element](#) can also be **muted**. If anything is muting the element, then it is muted. (For example, when the [direction of playback](#) is backwards, the element is muted.)

The **muted** IDL attribute must return the value to which it was last set. When a [media element](#) is created, if the element has a muted content attribute specified, then the muted IDL attribute should be set to true; otherwise, the user agents may set the value to the user's preferred value (e.g., remembering the last set value across sessions, on a per-site basis or otherwise). While the muted IDL attribute is set to true, the [media element](#) must be [muted](#).

Whenever either of the values that would be returned by the `volume` and `muted` IDL attributes change, the user agent must [queue a task](#) to [fire a simple event](#) named `volumechange` at the [media element](#).

An element's **effective media volume** is determined as follows:

1. If the user has indicated that the user agent is to override the volume of the element, then the element's [effective media volume](#) is the volume desired by the user. Abort these steps.
2. If the element's audio output is [muted](#), the element's [effective media volume](#) is zero. Abort these steps.
3. Let `volume` be the [playback volume](#) of the audio portions of the [media element](#), in range 0.0 (silent) to 1.0 (loudest).
4. The element's [effective media volume](#) is `volume`, interpreted relative to the range 0.0 to 1.0, with 0.0 being silent, and 1.0 being the loudest setting, values in between increasing in loudness. The range need not be linear. The loudest setting may be lower than the system's loudest possible setting; for example the user could have set a maximum volume.

The **muted** content attribute on [media elements](#) is a [boolean attribute](#) that controls the default state of the audio output of the [media resource](#), potentially overriding user preferences.

The **defaultMuted** IDL attribute must [reflect](#) the [muted](#) content attribute.

NOTE:

This attribute has no dynamic effect (it only controls the default state of the element).

EXAMPLE 432

This video (an advertisement) autoplays, but to avoid annoying users, it does so without sound, and allows the user to turn the sound on.

```
<video src="adverts.cgi?kind=video" controls autoplay loop muted></video>
```

4.7.14.13. Time ranges

Objects implementing the `TimeRanges` interface represent a list of ranges (periods) of time.

```
interface TimeRanges {
  readonly attribute unsigned long length;
  double start(unsigned long index);
  double end(unsigned long index);
};
```

This definition is non-normative. Implementation requirements are given below this definition.

***media* . length**

Returns the number of ranges in the object.

***time* = *media* . start(*index*)**

Returns the time for the start of the range with the given index.

Throws an `IndexSizeError` exception if the index is out of range.

***time* = *media* . end(*index*)**

Returns the time for the end of the range with the given index.

Throws an `IndexSizeError` exception if the index is out of range.

The **length** IDL attribute must return the number of ranges represented by the object.

The **start(*index*)** method must return the position of the start of the *index*th range represented by the object, in seconds measured from the start of the timeline that the object covers.

The **end(*index*)** method must return the position of the end of the *index*th range represented by the object, in seconds measured from the start of the timeline that the object covers.

These methods must throw `IndexSizeError` exceptions if called with an *index* argument greater than or equal to the number of ranges represented by the object.

When a `TimeRanges` object is said to be a **normalized `TimeRanges` object**, the ranges it represents must obey the following criteria:

- The start of a range must be greater than the end of all earlier ranges.
- The start of a range must be less than or equal to the end of that same range.

In other words, the ranges in such an object are ordered, don't overlap, and don't touch (adjacent ranges are folded into one bigger range). A range can be empty (referencing just a single moment in time), e.g., to indicate that only one frame is currently buffered in the case that the user agent has discarded the entire [media resource](#) except for the current frame, when a [media element](#) is paused.

Ranges in a `TimeRanges` object must be inclusive.

EXAMPLE 433

Thus, the end of a range would be equal to the start of a following adjacent (touching but not overlapping) range. Similarly, a range covering a whole timeline anchored at zero would have a start equal to zero and an end equal to the duration of the timeline.

The timelines used by the objects returned by the `buffered`, `seekable` and `played` IDL attributes of [media elements](#) must be that element's [media timeline](#).

4.7.14.14. The `TrackEvent` interface

```
[Constructor(DOMString type, optional TrackEventInit eventInitDict)]
interface TrackEvent : Event {
  readonly attribute (VideoTrack or AudioTrack or TextTrack)? track;
};

dictionary TrackEventInit : EventInit {
  (VideoTrack or AudioTrack or TextTrack)? track;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

`event` . `track`

Returns the track object ([TextTrack](#), [AudioTrack](#), or [VideoTrack](#)) to which the event relates.

The **`track`** attribute must return the value it was initialized to. When the object is created, this attribute must be initialized to null. It represents the context information for the event.

4.7.14.15. Event summary

This section is non-normative.

The following events fire on [media elements](#) as part of the processing model described above:

Event name	Interface	Fired when...	Preconditions
loadstart	Event	The user agent begins looking for media data , as part of the resource selection algorithm .	<code>networkState</code> equals <code>NETWORK_LOADING</code>
progress	Event	The user agent is fetching media data .	<code>networkState</code> equals <code>NETWORK_LOADING</code>
suspend	Event	The user agent is intentionally not currently fetching media data .	<code>networkState</code> equals <code>NETWORK_IDLE</code>
abort	Event	The user agent stops fetching the media data before it is completely downloaded, but not due to an error.	<code>error</code> is an object with the code <code>MEDIA_ERR_ABORTED</code> . <code>networkState</code> equals either <code>NETWORK_EMPTY</code> or <code>NETWORK_IDLE</code> , depending on when the download was aborted.
error	Event	An error occurs while fetching the media data or the type of the resource is not supported media format.	<code>error</code> is an object with the code <code>MEDIA_ERR_NETWORK</code> or higher. <code>networkState</code> equals either <code>NETWORK_EMPTY</code> or <code>NETWORK_IDLE</code> , depending on when the download was aborted.
emptied	Event	A media element whose <code>networkState</code> was previously not in the <code>NETWORK_EMPTY</code> state has just switched to that state (either because of a fatal error during load that's about to be reported, or because the <code>load()</code> method was invoked while the resource	<code>networkState</code> is <code>NETWORK_EMPTY</code> ; all the IDL attributes are in their initial states.

Event name	Interface	Fired when...	Preconditions
		selection algorithm was already running).	
stalled	Event	The user agent is trying to fetch media data , but data is unexpectedly not forthcoming.	networkState is NETWORK_LOADING.
loadedmetadata	Event	The user agent has just determined the duration and dimensions of the media resource and the text tracks are ready .	readyState is newly equal to HAVE_METADATA or greater for the first time.
loadeddata	Event	The user agent can render the media data at the current playback position for the first time.	readyState newly increased to HAVE_CURRENT_DATA or greater for the first time.
canplay	Event	The user agent can resume playback of the media data , but estimates that if playback were to be started now, the media resource could not be rendered at the current playback rate up to its end without having to stop for further buffering of content.	readyState newly increased to HAVE_FUTURE_DATA or greater.
canplaythrough	Event	The user agent estimates that if playback were to be started now, the media resource could be rendered at the current playback rate all the way to its end without having to stop for further buffering.	readyState is newly equal to HAVE_ENOUGH_DATA.
playing	Event	Playback is ready to start after having been paused or delayed due to lack of media data .	readyState is newly equal to or greater than HAVE_FUTURE_DATA and paused is false, or paused is newly false and readyState is equal to or greater than HAVE_FUTURE_DATA. Even if this event fires, the element might still not be potentially playing , e.g., if the element is paused for user interaction or paused for in-band content .
waiting	Event	Playback has stopped because the next frame is not available, but the user agent expects that frame to become available in due course.	readyState is equal to or less than HAVE_CURRENT_DATA, and paused is false. Either seeking is true, or the current playback position is not contained in any of the ranges in buffered. It is possible for playback to stop for other reasons without paused being false, but those reasons do not fire this event (and when those situations resolve, a separate playing event is not fired either): e.g., the playback ended , or playback stopped due to errors , or the element has paused for user interaction or paused for in-band content .
seeking	Event	The seeking IDL attribute changed to true, and the user agent has started seeking to a new position.	
seeked	Event	The seeking IDL attribute changed to false after the current playback position was changed.	
ended	Event	Playback has stopped because the end of the media resource was reached.	currentTime equals the end of the media resource ; ended is true.
durationchange	Event	The duration attribute has just been updated.	

Event name	Interface	Fired when...	Preconditions
timeupdate	Event	The current playback position changed as part of normal playback or in an especially interesting way, for example discontinuously.	
play	Event	The element is no longer paused. Fired after the <code>play()</code> method has returned, or when the <code>autoplay</code> attribute has caused playback to begin.	<code>paused</code> is newly false.
pause	Event	The element has been paused. Fired after the <code>pause()</code> method has returned.	<code>paused</code> is newly true.
ratechange	Event	Either the <code>defaultPlaybackRate</code> or the <code>playbackRate</code> attribute has just been updated.	
resize	Event	One or both of the <code>videoWidth</code> and <code>videoHeight</code> attributes have just been updated.	Media element is a <video> element; <code>readyState</code> is not <code>HAVE_NOTHING</code>
volumechange	Event	Either the <code>volume</code> attribute or the <code>muted</code> attribute has changed. Fired after the relevant attribute's setter has returned.	

The following event fires on [<source>](#) element:

Event name	Interface	Fired when...
error	Event	An error occurs while fetching the media data or the type of the resource is not supported media format.

The following events fire on [AudioTrackList](#), [VideoTrackList](#), and [TextTrackList](#) objects:

Event name	Interface	Fired when...
change	Event	One or more tracks in the track list have been enabled or disabled.
addtrack	TrackEvent	A track has been added to the track list.
removetrack	TrackEvent	A track has been removed from the track list.

The following event fires on [TextTrack](#) objects and [<track>](#) elements:

Event name	Interface	Fired when...
cuechange	Event	One or more cues in the track have become active or stopped being active.

The following events fire on track elements:

Event name	Interface	Fired when...
error	Event	An error occurs while fetching the track data or the type of the resource is not supported text track format.
load	Event	A track data has been fetched and successfully processed.

The following events fire on [TextTrackCue](#) objects:

Event name	Interface	Fired when...
enter	Event	The cue has become active.
exit	Event	The cue has stopped being active.

§ 4.7.14.16. Security and privacy considerations

The main security and privacy implications of the video and audio elements come from the ability to embed media cross-origin. There are two directions that threats can flow: from hostile content to a victim page, and from a hostile page to victim content.



If a victim page embeds hostile content, the threat is that the content might contain scripted code that attempts to interact with the [Document](#) that embeds the content. To avoid this, user agents must ensure that there is no access from the content to the embedding page. In the case of media content that uses DOM concepts, the embedded content must be treated as if it was in its own unrelated [top-level browsing context](#).

EXAMPLE 434

For instance, if an SVG animation was embedded in a [<video>](#) element, the user agent would not give it access to the DOM of the outer page. From the perspective of scripts in the SVG resource, the SVG file would appear to be in a lone top-level browsing context with no parent.



If a hostile page embeds victim content, the threat is that the embedding page could obtain information from the content that it would not otherwise have access to. The API does expose some information: the existence of the media, its type, its duration, its size, and the performance characteristics of its host. Such information is already potentially problematic, but in practice the same information can be obtained using the [](#) element, and so it has been deemed acceptable.

However, significantly more sensitive information could be obtained if the user agent further exposes metadata within the content such as subtitles or chapter titles. Such information is therefore only exposed if the video resource passes a CORS resource sharing check. The `crossorigin` attribute allows authors to control how this check is performed. [\[FETCH\]](#)

EXAMPLE 435

Without this restriction, an attacker could trick a user running within a corporate network into visiting a site that attempts to load a video from a previously leaked location on the corporation's intranet. If such a video included confidential plans for a new product, then being able to read the subtitles would present a serious confidentiality breach.

§ 4.7.14.17. Best practices for authors using media elements

This section is non-normative.

Playing audio and video resources on small devices such as set-top boxes or mobile phones is often constrained by limited hardware resources in the device. For example, a device might only support three simultaneous videos. For this reason, it is a good practice to release resources held by [media elements](#) when they are done playing, either by being very careful about removing all references to the element and allowing it to be garbage collected, or, even better, by removing the element's `src` attribute and any [<source>](#) element descendants, and invoking the element's `load()` method.

Similarly, when the playback rate is not exactly 1.0, hardware, software, or format limitations can cause video frames to be dropped and audio to be choppy or muted.

§ 4.7.14.18. Best practices for implementors of media elements

This section is non-normative.

How accurately various aspects of the [media element](#) API are implemented is considered a quality-of-implementation issue.

For example, when implementing the `buffered` attribute, how precise an implementation reports the ranges that have been buffered depends on how carefully the user agent inspects the data. Since the API reports ranges as times, but the data is obtained in byte streams, a user agent receiving a variable-bit-rate stream might only be able to determine precise times by actually decoding all of the data. User agents aren't required to do this, however; they can instead return estimates (e.g., based

on the average bitrate seen so far) which get revised as more information becomes available.

As a general rule, user agents are urged to be conservative rather than optimistic. For example, it would be bad to report that everything had been buffered when it had not.

Another quality-of-implementation issue would be playing a video backwards when the codec is designed only for forward playback (e.g., there aren't many key frames, and they are far apart, and the intervening frames only have deltas from the previous frame). User agents could do a poor job, e.g., only showing key frames; however, better implementations would do more work and thus do a better job, e.g., actually decoding parts of the video forwards, storing the complete frames, and then playing the frames backwards.

Similarly, while implementations are allowed to drop buffered data at any time (there is no requirement that a user agent keep all the media data obtained for the lifetime of the media element), it is again a quality of implementation issue: user agents with sufficient resources to keep all the data around are encouraged to do so, as this allows for a better user experience. For example, if the user is watching a live stream, a user agent could allow the user only to view the live video; however, a better user agent would buffer everything and allow the user to seek through the earlier material, pause it, play it forwards and backwards, etc.



When a [media element](#) that is paused is [removed from a document](#) and not reinserted before the next time the [event loop](#) reaches step 1, implementations that are resource constrained are encouraged to take that opportunity to release all hardware resources (like video planes, networking resources, and data buffers) used by the [media element](#). (User agents still have to keep track of the playback position and so forth, though, in case playback is later restarted.)

§ 4.7.15. The `map` element

Categories:

[Flow content.](#)
[Phrasing content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Transparent.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

name - Name of [image map](#) to reference from the usemap attribute

Allowed ARIA role attribute values:

None

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

DOM interface:

```
interface HTMLMapElement : HTMLElement {
  attribute DOMString name;
  [SameObject] readonly attribute HTMLCollection areas;
  [SameObject] readonly attribute HTMLCollection images;
};
```

The `<map>` element, in conjunction with an `<img>` element and any `<area>` element descendants, defines an [image map](#). The element [represents](#) its children.

The **name** attribute gives the map a name so that it can be referenced. The attribute must be present and must have a non-empty value with no [space characters](#). The value of the name attribute must not be a [compatibility caseless](#) match for the value of the name attribute of another [map](#) element in the same document. If the id attribute is also specified, both attributes must have the same value.

This definition is non-normative. Implementation requirements are given below this definition.

map . areas

Returns an HTMLCollection of the [area](#) elements in the [map](#).

map . images

Returns an HTMLCollection of the [img](#) and [object](#) elements that use the [map](#).

The **areas** attribute must return an HTMLCollection rooted at the [map](#) element, whose filter matches only [area](#) elements.

The **images** attribute must return an HTMLCollection rooted at the [Document](#) node, whose filter matches only [img](#) and [object](#) elements that are associated with this map element according to the [image map](#) processing model.

The IDL attribute **name** must [reflect](#) the content attribute of the same name.

EXAMPLE 436

Image maps can be defined in conjunction with other content on the page, to ease maintenance. This example is of a page with an image map at the top of the page and a corresponding set of text links at the bottom.

```
<!DOCTYPE HTML>
<TITLE>Babies™: Toys</TITLE>
<HEADER>
  <h1>Toys</h1>
  <IMG SRC="/images/menu.gif"
    ALT="Babies™ navigation menu. Select a department to go to its page."
    USEMAP="#NAV">
</HEADER>
...
<FOOTER>
  <MAP NAME="NAV">
    <P>
      <A HREF="/clothes/">Clothes</A>
      <AREA ALT="Clothes" COORDS="0,0,100,50" HREF="/clothes/"> |
      <A HREF="/toys/">Toys</A>
      <AREA ALT="Toys" COORDS="100,0,200,50" HREF="/toys/"> |
      <A HREF="/food/">Food</A>
      <AREA ALT="Food" COORDS="200,0,300,50" HREF="/food/"> |
      <A HREF="/books/">Books</A>
      <AREA ALT="Books" COORDS="300,0,400,50" HREF="/books/">
    </P>
  </MAP>
</FOOTER>
```

§ 4.7.16. The area element

Categories:

[Flow content.](#)

[Phrasing content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected, but only if there is a [map](#) element ancestor or a [template](#) element ancestor.

Content model:

[Nothing.](#)

Tag omission in text/html:

No [end tag](#)

Content attributes:

Global attributes

alt - Replacement text for use when images are not available

coords - Coordinates for the shape to be created in an [image map](#)

download - Whether to download the resource instead of navigating to it, and its file name if so

href - Address of the [hyperlink](#)

hreflang - Language of the linked resource

rel - Relationship of this document (or subsection/topic) to the destination resource

shape - The kind of shape to be created in an [image map](#)

target - [browsing context](#) for [hyperlink navigation](#)

type - Hint for the type of the referenced resource

Allowed [ARIA role attribute](#) values:

'[link](#)' role (default - *do not set*).

Allowed [ARIA state and property attributes](#):

Global aria-* attributes

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLAreaElement : HTMLElement {
  attribute DOMString alt;
  attribute DOMString coords;
  attribute DOMString shape;
  attribute DOMString target;
  attribute DOMString download;
  attribute DOMString rel;
  readonly attribute DOMTokenList relList;
  attribute DOMString hreflang;
  attribute DOMString type;
};
HTMLAreaElement implements HTMLHyperlinkElementUtils;
```

The `<area>` element [represents](#) either a hyperlink with some text and a corresponding area on an [image map](#), or a dead area on an image map.

An `<area>` element with a parent node must have a `<map>` element ancestor or a `<template>` element ancestor.

If the `<area>` element has an `href` attribute, then the `<area>` element represents a [hyperlink](#). In this case, the **alt** attribute must be present. It specifies the text of the hyperlink. Its value must be text that informs the user about the destination of the link.

If the `<area>` element has no `href` attribute, then the area represented by the element cannot be selected, and the **alt** attribute must be omitted.

In both cases, the **shape** and **coords** attributes specify the area.

The **shape** attribute is an [enumerated attribute](#). The following table lists the keywords defined for this attribute. The states given in the first cell of the rows with keywords give the states to which those keywords map. Some of the keywords are non-conforming, as noted in the last column.

State	Keywords	Notes
Circle state	circle	
	circ	Non-conforming
Default state	default	
Polygon state	poly	

State	Keywords	Notes
	polygon	Non-conforming
Rectangle state	rect	
	rectangle	Non-conforming

The attribute may be omitted. The *missing value default* is the [rectangle state](#).

The **coords** attribute must, if specified, contain a [valid list of floating-point numbers](#). This attribute gives the coordinates for the shape described by the shape attribute. The processing for this attribute is described as part of the [image map](#) processing model.

In the **circle state**, [area](#) elements must have a coords attribute present, with three integers, the last of which must be non-negative. The first integer must be the distance in CSS pixels from the left edge of the image to the center of the circle, the second integer must be the distance in CSS pixels from the top edge of the image to the center of the circle, and the third integer must be the radius of the circle, again in CSS pixels.

In the **default state** state, [area](#) elements must not have a coords attribute. (The area is the whole image.)

In the **polygon state**, [area](#) elements must have a coords attribute with at least six integers, and the number of integers must be even. Each pair of integers must represent a coordinate given as the distances from the left and the top of the image in CSS pixels respectively, and all the coordinates together must represent the points of the polygon, in order.

In the **rectangle state**, [area](#) elements must have a coords attribute with exactly four integers, the first of which must be less than the third, and the second of which must be less than the fourth. The four points must represent, respectively, the distance from the left edge of the image to the left side of the rectangle, the distance from the top edge to the top side, the distance from the left edge to the right side, and the distance from the top edge to the bottom side, all in CSS pixels.

When user agents allow users to [follow hyperlinks](#) or [download hyperlinks](#) created using the [area](#) element, as described in the next section, the href, target, and download attributes decide how the link is followed. The [rel](#), hreflang, and type attributes may be used to indicate to the user the likely nature of the target resource before the user follows the link.

The target, download, [rel](#), hreflang, and type attributes must be omitted if the href attribute is not present.

The [activation behavior](#) of [area](#) elements is to run the following steps:

1. If the [area](#) element's [node document](#) is not [fully active](#), then abort these steps.
2. If the [area](#) element has a download attribute and the algorithm is not [allowed to show a popup](#); or, if the user has not indicated a specific [browsing context](#) for following the link, and the element's target attribute is present, and applying [the rules for choosing a browsing context given a browsing context name](#), using the value of the target attribute as the browsing context name, would result in there not being a chosen browsing context, then run these substeps:
 1. If there is an [entry settings object](#), throw an `InvalidAccessError` exception.
 2. Abort these steps without following the hyperlink.
3. Otherwise, the user agent must [follow the hyperlink](#) or [download the hyperlink](#) created by the [area](#) element, if any, and as determined by the download attribute and any expressed user preference.

The IDL attributes **alt**, **coords**, **target**, **download**, **rel**, **hreflang**, and **type**, each must [reflect](#) the respective content attributes of the same name.

The IDL attribute **shape** must [reflect](#) the [shape](#) content attribute.

The IDL attribute **relList** must [reflect](#) the [rel](#) content attribute.



The [area](#) element also supports the `HTMLHyperlinkElementUtils` interface. [\[URL\]](#)

When the element is created, and whenever the element's href content attribute is set, changed, or removed, the user agent must invoke the element's `HTMLHyperlinkElementUtils` interface's `set the input algorithm` with the value of the href con-

tent attribute, if any, or the empty string otherwise, as the given value.

The element's `HTMLHyperlinkElementUtils` interface's `get` the base algorithm must simply return the [document base URL](#).

The element's `HTMLHyperlinkElementUtils` interface's `query` encoding is the [document's character encoding](#).

When the element's `HTMLHyperlinkElementUtils` interface invokes its `update` steps with a string `value`, the user agent must set the element's `href` content attribute to the string `value`.

§ 4.7.17. Image maps

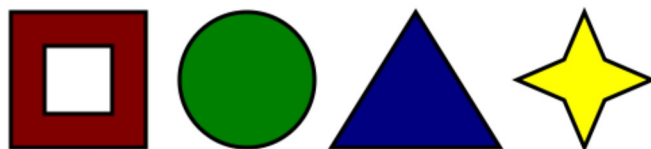
§ 4.7.17.1. Authoring

An **image map** allows geometric areas on an image to be associated with [hyperlinks](#).

An image, in the form of an `<img>` element, may be associated with an image map (in the form of a `map` element) by specifying a `usemap` attribute on the `<img>` element. The `usemap` attribute, if specified, must be a [valid hash-name reference](#) to a `<map>` element.

EXAMPLE 437

Consider an image that looks as follows:



If we wanted just the colored areas to be clickable, we could do it as follows:

```
<p>
  Please select a shape:
  
  <map name="shapes">
    <area shape=rect coords="50,50,100,100"> <!-- the hole in the red box -->
    <area shape=rect coords="25,25,125,125" href="red.html" alt="Red box.">
    <area shape=circle coords="200,75,50" href="green.html" alt="Green circle.">
    <area shape=poly coords="325,25,262,125,388,125" href="blue.html" alt="Blue triangle.">
    <area shape=poly coords="450,25,435,60,400,75,435,90,450,125,465,90,500,75,465,60"
      href="yellow.html" alt="Yellow star.">
  </map>
</p>
```

§ 4.7.17.2. Processing model

If an `<img>` element has a `usemap` attribute specified, user agents must process it as follows:

1. Parse the attribute's value using the [rules for parsing a hash-name reference](#) to a `<map>` element, with the element's [node document](#) as the context node. This will return either an element (the `map`) or null.
2. If that returned null, then abort these steps. The image is not associated with an image map after all.
3. Otherwise, the user agent must collect all the `<area>` elements that are descendants of the `map`. Let those be the `areas`.

Having obtained the list of `<area>` elements that form the image map (the `areas`), interactive user agents must process the list in one of two ways.

If the user agent intends to show the text that the `<img>` element represents, then it must use the following steps.

NOTE:

In user agents that do not support images, or that have images disabled, `<object>` elements cannot represent images, and thus this section never applies (the [fallback content](#) is shown instead). The following steps therefore only apply to `<img>` elements.

1. Remove all the `<area>` elements in `areas` that have no href attribute.
2. Remove all the `<area>` elements in `areas` that have no alt attribute, or whose alt attribute's value is the empty string, if there is another `<area>` element in `areas` with the same value in the href attribute and with a non-empty alt attribute.
3. Each remaining `<area>` element in `areas` represents a [hyperlink](#). Those hyperlinks should all be made available to the user in a manner associated with the text of the `<img>`.

In this context, user agents may represent area and `<img>` elements with no specified alt attributes, or whose alt attributes are the empty string or some other non-visible text, in a user-agent-defined fashion intended to indicate the lack of suitable author-provided text.

If the user agent intends to show the image and allow interaction with the image to select hyperlinks, then the image must be associated with a set of layered shapes, taken from the `<area>` elements in `areas`, in reverse [tree order](#) (so the last specified `<area>` element in the `map` is the bottom-most shape, and the first element in the `map`, in [tree order](#), is the top-most shape).

Each `<area>` element in `areas` must be processed as follows to obtain a shape to layer onto the image:

1. Find the state that the element's shape attribute represents.
2. Use the [rules for parsing a list of floating-point numbers](#) to parse the element's coords attribute, if it is present, and let the result be the `coords` list. If the attribute is absent, let the `coords` list be the empty list.
3. If the number of items in the `coords` list is less than the minimum number given for the `<area>` element's current state, as per the following table, then the shape is empty; abort these steps.

State	Minimum number of items
Circle state	3
Default state	0
Polygon state	6
Rectangle state	4

4. Check for excess items in the `coords` list as per the entry in the following list corresponding to the shape attribute's state:

↪ [Circle state](#)

Drop any items in the list beyond the third.

↪ [Default state](#)

Drop all items in the list.

↪ [Polygon state](#)

Drop the last item if there's an odd number of items.

↪ [Rectangle state](#)

Drop any items in the list beyond the fourth.

5. If the shape attribute represents the [rectangle state](#), and the first number in the list is numerically greater than the third number in the list, then swap those two numbers around.
6. If the shape attribute represents the [rectangle state](#), and the second number in the list is numerically greater than the fourth number in the list, then swap those two numbers around.

7. If the shape attribute represents the [circle state](#), and the third number in the list is less than or equal to zero, then the shape is empty; abort these steps.
8. Now, the shape represented by the element is the one described for the entry in the list below corresponding to the state of the shape attribute:

↪ [Circle state](#)

Let x be the first number in `coords`, y be the second number, and r be the third number.

The shape is a circle whose center is x CSS pixels from the left edge of the image and y CSS pixels from the top edge of the image, and whose radius is r pixels.

↪ [Default state](#)

The shape is a rectangle that exactly covers the entire image.

↪ [Polygon state](#)

Let x_i be the $(2i)$ th entry in `coords`, and y_i be the $(2i+1)$ th entry in `coords` (the first entry in `coords` being the one with index 0).

Let *the coordinates* be (x_i, y_i) , interpreted in CSS pixels measured from the top left of the image, for all integer values of i from 0 to $(N/2)-1$, where N is the number of items in `coords`.

The shape is a polygon whose vertices are given by *the coordinates*, and whose interior is established using the even-odd rule. [\[GRAPHICS\]](#)

↪ [Rectangle state](#)

Let x_1 be the first number in `coords`, y_1 be the second number, x_2 be the third number, and y_2 be the fourth number.

The shape is a rectangle whose top-left corner is given by the coordinate (x_1, y_1) and whose bottom right corner is given by the coordinate (x_2, y_2) , those coordinates being interpreted as CSS pixels from the top left corner of the image.

For historical reasons, the coordinates must be interpreted relative to the *displayed* image after any stretching caused by the CSS `'width'` and `'height'` properties (or, for non-CSS browsers, the image element's `width` and `height` attributes — CSS browsers map those attributes to the aforementioned CSS properties).

NOTE:

Browser zoom features and transforms applied using CSS or SVG do not affect the coordinates.

Pointing device interaction with an image associated with a set of layered shapes per the above algorithm must result in the relevant user interaction events being first fired to the top-most shape covering the point that the pointing device indicated, if any, or to the image element itself, if there is no shape covering that point. User agents may also allow individual [area](#) elements representing [hyperlinks](#) to be selected and activated (e.g., using a keyboard).

NOTE:

Because a [map](#) element (and its [area](#) elements) can be associated with multiple [img](#) and [object](#) elements, it is possible for an [area](#) element to correspond to multiple focusable areas of the document.

Image maps are [live](#); if the DOM is mutated, then the user agent must act as if it had rerun the algorithms for image maps.

§ 4.7.18. MathML

The [math](#) element from the [MathML namespace](#) falls into the [embedded content](#), [phrasing content](#), [flow content](#), and [paleable content](#) categories for the purposes of the content models in this specification.

This specification refers to several specific MathML elements, in particular: [<annotation-xml>](#), [<merror>](#), [<mi>](#), [<mn>](#), [<mo>](#), [<ms>](#), and [<mtext>](#).

When the MathML [<annotation-xml>](#) element contains elements from the [HTML namespace](#), such elements must all be [flow content](#). [\[MATHML\]](#)

When the MathML token elements (`<mi>`, `<mo>`, `<mn>`, `<ms>`, and `<mtext>`) are descendants of HTML elements, they may contain [phrasing content](#) elements from the [HTML namespace](#). [\[MATHML\]](#)

User agents must handle text other than [inter-element whitespace](#) found in MathML elements whose content models do not allow straight text by pretending for the purposes of MathML content models, layout, and rendering that the text is actually wrapped in an `<mtext>` element in the [MathML namespace](#). (Such text is not, however, conforming.)

User agents must act as if any MathML element whose contents does not match the element's content model was replaced, for the purposes of MathML layout and rendering, by an `<merror>` element in the [MathML namespace](#) containing some appropriate error message.

To enable authors to use MathML tools that only accept MathML in its XML form, interactive HTML user agents are encouraged to provide a way to export any MathML fragment as an XML namespace-well-formed XML fragment.

The semantics of MathML elements are defined by the MathML specification and [other applicable specifications](#). [\[MATHML\]](#)

EXAMPLE 438

Here is an example of the use of MathML in an HTML document:

```
<!DOCTYPE html>
<html>
  <head>
    <title>The quadratic formula</title>
  </head>
  <body>
    <h1>The quadratic formula</h1>
    <p>
      <math>
        <mi>x</mi>
        <mo>=</mo>
        <mfrac>
          <mrow>
            <mo form="prefix">-</mo> <mi>b</mi>
            <mo>±</mo>
            <msqrt>
              <msup> <mi>b</mi> <mn>2</mn> </msup>
              <mo>-</mo>
              <mn>4</mn> <mo></mo> <mi>a</mi> <mo></mo> <mi>c</mi>
            </msqrt>
          </mrow>
          <mrow>
            <mn>2</mn> <mo></mo> <mi>a</mi>
          </mrow>
        </mfrac>
      </math>
    </p>
  </body>
</html>
```

§ 4.7.19. SVG

The `svg` element from the [SVG namespace](#) falls into the [embedded content](#), [phrasing content](#), [flow content](#), and [palpable content](#) categories for the purposes of the content models in this specification.

To enable authors to use SVG tools that only accept SVG in its XML form, interactive HTML user agents are encouraged to provide a way to export any SVG fragment as an XML namespace-well-formed XML fragment.

When the SVG `<foreignObject>` element contains elements from the [HTML namespace](#), such elements must all be [flow content](#). [\[SVG11\]](#)

The content model for `<title>` elements in the [SVG namespace](#) inside [HTML documents](#) is [phrasing content](#). (This further constrains the requirements given in the SVG specification.)

The semantics of SVG elements are defined by the SVG specification and [other applicable specifications](#). [\[SVG11\]](#)

User agent requirements: SVG as implemented today follows neither SVG 1.1 nor SVG Tiny 1.2 precisely, instead implementing subsets of each. Although it is hoped that the in-progress SVG 2 specification is a more realistic target for implementations, until that specification is ready, user agents must implement the SVG 1.1 specification with the following willful violations and additions. [\[SVG11\]](#) [\[SVGTiny12\]](#) [\[SVG2\]](#)

The following features from SVG 1.1 must not be implemented:

- The `tref` element
- The `cursor` element (use CSS's `cursor` property instead)
- The font-defining elements: `font`, `glyph`, `missing-glyph`, `hkern`, `vkern`, `font-face`, `font-face-src`, `font-face-uri`, `font-face-format`, and `font-face-name` (use CSS's `@font-face` instead)
- The `externalResourcesRequired` attribute
- The `enable-background` property
- The `contentType` and `styleType` attributes (use the `type` attribute on the `script` and `style` elements instead)

The following features from SVG Tiny 1.2 must be implemented:

- The `non-scaling-stroke` value for the `vector-effect` property
- The `class` attribute is allowed on all SVG elements
- The `tabindex` attribute is allowed on visible SVG elements
- The ARIA accessibility attributes are allowed on all SVG elements

§ 4.7.20. Dimension attributes

Author requirements: The `width` and `height` attributes on `<img>`, `<iframe>`, `<embed>`, `<object>`, `<video>`, and, when their type attribute is in the [image button](#) state, `<input>` elements may be specified to give the dimensions of the visual content of the element (the width and height respectively, relative to the nominal direction of the output medium), in CSS pixels. The attributes, if specified, must have values that are [valid non-negative integers](#).

The specified dimensions given may differ from the dimensions specified in the resource itself, since the resource may have a resolution that differs from the CSS pixel resolution. (On screens, CSS pixels have a resolution of 96ppi, but in general the CSS pixel resolution depends on the reading distance.) If both attributes are specified, then one of the following statements must be true:

- $\text{specified width} - 0.5 \leq \text{specified height} * \text{target ratio} \leq \text{specified width} + 0.5$
- $\text{specified height} - 0.5 \leq \text{specified width} / \text{target ratio} \leq \text{specified height} + 0.5$
- $\text{specified height} = \text{specified width} = 0$

The `target ratio` is the ratio of the [intrinsic width](#) to the [intrinsic height](#) in the resource. The `specified width` and `specified height` are the values of the width and height attributes respectively.

The two attributes must be omitted if the resource in question does not have both an [intrinsic width](#) and an [intrinsic height](#).

If the two attributes are both zero, it indicates that the element is not intended for the user (e.g., it might be a part of a service to count page views).

NOTE:

The dimension attributes are not intended to be used to stretch the image.

User agent requirements: User agents are expected to use these attributes [as hints for the rendering](#).

The **width** and **height** IDL attributes on the `<iframe>`, `<embed>`, `<object>`, and `<video>` elements must [reflect](#) the respective content attributes of the same name.

NOTE:

For `<iframe>`, `<embed>`, and `<object>` the IDL attributes are DOMString; for `<video>` the IDL attributes are unsigned long.

NOTE:

The corresponding IDL attributes for `<img>` and `<input>` elements are defined in those respective elements' sections, as they are slightly more specific to those elements' other behaviors.

§ 4.8. Links

§ 4.8.1. Introduction

Links are a conceptual construct, created by `<a>`, `<area>`, and `<link>` elements, that [represent](#) a connection between two resources, one of which is the current [Document](#). There are two kinds of links in HTML:

Links to external resources

These are links to resources that are to be used to augment the current document, generally automatically processed by the user agent.

Hyperlinks

These are links to other resources that are generally exposed to the user by the user agent so that the user can cause the user agent to [navigate](#) to those resources, e.g., to visit them in a browser or download them.

For `<link>` elements with an `href` attribute and a `rel` attribute, links must be created for the keywords of the `rel` attribute, as defined for those keywords in the [link types](#) section.

Similarly, for `<a>` and `<area>` elements with an `href` attribute and a `rel` attribute, links must be created for the keywords of the `rel` attribute as defined for those keywords in the [link types](#) section. Unlike `<link>` elements, however, `<a>` and `<area>` elements with an `href` attribute that either do not have a `rel` attribute, or whose `rel` attribute has no keywords that are defined as specifying [hyperlinks](#), must also create a [hyperlink](#). This implied hyperlink has no special meaning (it has no [link type](#)) beyond linking the element's [node document](#) to the resource given by the element's `href` attribute.

A [hyperlink](#) can have one or more **hyperlink annotations** that modify the processing semantics of that hyperlink.

§ 4.8.2. Links created by `<a>` and `<area>` elements

The `href` attribute on `<a>` and `<area>` elements must have a value that is a [valid URL potentially surrounded by spaces](#).

NOTE:

The `href` attribute on `<a>` and `<area>` elements is not required; when those elements do not have `href` attributes they do not create hyperlinks.

The `target` attribute, if present, must be a [valid browsing context name or keyword](#). It gives the name of the [browsing context](#) that will be used. User agents use this name when [following hyperlinks](#).

When an `<a>` or `<area>` element's [activation behavior](#) is invoked, the user agent may allow the user to indicate a preference regarding whether the hyperlink is to be used for [navigation](#) or whether the resource it specifies is to be downloaded.

In the absence of a user preference, the default should be navigation if the element has no `download` attribute, and should be to download the specified resource if it does.

Whether determined by the user's preferences or via the presence or absence of the attribute, if the decision is to use the hyperlink for [navigation](#) then the user agent must [follow the hyperlink](#), and if the decision is to use the hyperlink to download a resource, the user agent must [download the hyperlink](#). These terms are defined in subsequent sections below.

The **download** attribute, if present, indicates that the author intends the hyperlink to be used for downloading a resource. The attribute may have a value; the value, if any, specifies the default file name that the author recommends for use in labeling the resource in a local file system. There are no restrictions on allowed values, but authors are cautioned that most file systems have limitations with regard to what punctuation is supported in file names, and user agents are likely to adjust file names accordingly.

The **rel** attribute on `<a>` and `<area>` elements controls what kinds of links the elements create. The attribute's value must be a [set of space-separated tokens](#). The [allowed keywords and their meanings](#) are defined below.

[rel's supported tokens](#) are the keywords defined in [HTML link types](#) which are allowed on `<a>` and `<area>` elements, impact the processing model, and are supported by the user agent. The possible supported tokens are `noreferrer`, and `noopener`. [rel's supported tokens](#) must only include the tokens from this list that the user agent implements the processing model for.

Other specifications may add [HTML link types](#) as defined in [Other link types](#). These specifications may require that their link types be included in [rel](#)'s supported tokens.

The [rel](#) attribute has no default value. If the attribute is omitted or if none of the values in the attribute are recognized by the user agent, then the document has no particular relationship with the destination resource other than there being a hyperlink between the two.

The **hreflang** attribute on `<a>` elements that create [hyperlinks](#), if present, gives the language of the linked resource. It is purely advisory. The value must be a valid BCP 47 language tag. [\[BCP47\]](#) User agents must not consider this attribute authoritative — upon fetching the resource, user agents must use only language information associated with the resource to determine its language, not metadata included in the link to the resource.

The **type** attribute, if present, gives the [MIME type](#) of the linked resource. It is purely advisory. The value must be a [valid mime type](#). User agents must not consider the **type** attribute authoritative — upon fetching the resource, user agents must not use metadata included in the link to the resource to determine its type.

§ 4.8.3. API for `<a>` and `<area>` elements

```
[NoInterfaceObject]
interface HTMLHyperlinkElementUtils {
  stringifier attribute USVString href;
  readonly attribute USVString origin;
  attribute USVString protocol;
  attribute USVString username;
  attribute USVString password;
  attribute USVString host;
  attribute USVString hostname;
  attribute USVString port;
  attribute USVString pathname;
  attribute USVString search;
  attribute USVString hash;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

`hyperlink . toString()`

`hyperlink . href`

Returns the hyperlink's URL.

Can be set, to change the URL.

`hyperlink . origin`

Returns the hyperlink's URL's origin.

`hyperlink . protocol`

Returns the hyperlink's URL's scheme.

Can be set, to change the URL's scheme.

`hyperlink . username`

Returns the hyperlink's URL's username.

Can be set, to change the URL's username.

***hyperlink* . password**

Returns the hyperlink's URL's password.

Can be set, to change the URL's password.

***hyperlink* . host**

Returns the hyperlink's URL's host and port (if different from the default port for the scheme).

Can be set, to change the URL's host and port.

***hyperlink* . hostname**

Returns the hyperlink's URL's host.

Can be set, to change the URL's host.

***hyperlink* . port**

Returns the hyperlink's URL's port.

Can be set, to change the URL's port.

***hyperlink* . pathname**

Returns the hyperlink's URL's path.

Can be set, to change the URL's path.

***hyperlink* . search**

Returns the hyperlink's URL's query (includes leading "?" if non-empty).

Can be set, to change the URL's query (ignores leading "?").

***hyperlink* . hash**

Returns the hyperlink's URL's fragment (includes leading "#" if non-empty).

Can be set, to change the URL's fragment (ignores leading "#").

An element implementing the `HTMLHyperlinkElementUtils` mixin has an associated url (null or a [URL](#)). It is initially null.

An element implementing the `HTMLHyperlinkElementUtils` mixin has an associated **set the url** algorithm, which sets this element's [URL](#) to the [resulting URL string](#) of [parsing](#) this element's href content attribute value relative to this element. If [parsing](#) was aborted with an error, set this element's [URL](#) to null.

When elements implementing the `HTMLHyperlinkElementUtils` mixin are created, and whenever those elements have their href content attribute set, changed, or removed, the user agent must [set the url](#).

NOTE:

This is only observable for blob: URLs as [parsing](#) them involves the [StructuredClone](#) abstract algorithm.

An element implementing the `HTMLHyperlinkElementUtils` mixin has an associated **reinitialise url** algorithm, which runs these steps:

1. If element's [URL](#) is non-null, its [scheme](#) is "blob", and its [non-relative flag](#) is set, terminate these steps.
2. [Set the url](#).

To **update href**, set the element's href content attribute's value to the element's [URL](#), serialized.



The **href** attribute's getter must run these steps:

1. [Reinitialise url](#).

2. Let `url` be this element's [URL](#).
3. If `url` is null and this element has no `href` content attribute, return the empty string.
4. Otherwise, if `url` is null, return this element's `href` content attribute's value.
5. Return `url`, serialized.

The `href` attribute's setter must set this element's `href` content attribute's value to the given value.

The **origin** attribute's getter must run these steps:

1. [Reinitialise url](#).
2. If this element's [URL](#) is null, return the empty string.
3. Return the [Unicode serialization](#) of this element's [URL](#)'s [origin](#).

NOTE:

It returns the Unicode rather than the ASCII serialization for compatibility with `MessageEvent`.

The **protocol** attribute's getter must run these steps:

1. [Reinitialise url](#).
2. If this element's [URL](#) is null, return ":".
3. Return this element's [URL](#)'s [scheme](#), followed by ":".

The `protocol` attribute's setter must run these steps:

1. [Reinitialise url](#).
2. If this element's [URL](#) is null, terminate these steps.
3. [Basic URL parse](#) the given value, followed by ":", with this element's [URL](#) as `url` and [scheme start state](#) as *state override*.
4. [Update href](#).

The **username** attribute's getter must run these steps:

1. [Reinitialise url](#).
2. If this element's [URL](#) is null, return the empty string.
3. Return this element's [URL](#)'s [username](#).

The `username` attribute's setter must run these steps:

1. [Reinitialise url](#).
2. Let `url` be this element's [URL](#).
3. If `url` or `url`'s [host](#) is null, or `url`'s [non-relative flag](#) is set, terminate these steps.
4. [set the username](#), given `url` and the given value.
5. [Update href](#).

The **password** attribute's getter must run these steps:

1. [Reinitialise url](#).
2. Let `url` be this element's [URL](#).
3. If `url` or `url`'s [password](#) is null, return the empty string.

4. Return `url`'s [password](#).

The `password` attribute's setter must run these steps:

1. [Reinitialise url](#).
2. Let `url` be this element's [URL](#).
3. If `url` or `url`'s [host](#) is null, or `url`'s [non-relative flag](#) is set, terminate these steps.
4. [Set the password](#), given `url` and the given value.
5. [Update href](#).

The `host` attribute's getter must run these steps:

1. [Reinitialise url](#).
2. Let `url` be this element's [URL](#).
3. If `url` or `url`'s [host](#) is null, return the empty string.
4. If `url`'s [port](#) is null, return `url`'s [host](#), serialized.
5. Return `url`'s [host](#), serialized, followed by ":" and `url`'s [port](#), serialized.

The `host` attribute's setter must run these steps:

1. [Reinitialise url](#).
2. Let `url` be this element's [URL](#).
3. If `url` is null or `url`'s [non-relative flag](#) is set, terminate these steps.
4. [Basic URL parse](#) the given value, with `url` as `url` and `host state` as `state override`.
5. [Update href](#).

The `hostname` attribute's getter must run these steps:

1. [Reinitialise url](#).
2. Let `url` be this element's [URL](#).
3. If `url` or `url`'s [host](#) is null, return the empty string.
4. Return `url`'s [host](#), serialized.

The `hostname` attribute's setter must run these steps:

1. [Reinitialise url](#).
2. Let `url` be this element's [URL](#).
3. If `url` is null or `url`'s [non-relative flag](#) is set, terminate these steps.
4. [Basic URL parse](#) the given value, with `url` as `url` and `hostname state` as `state override`.
5. [Update href](#).

The `port` attribute's getter must run these steps:

1. [Reinitialise url](#).
2. Let `url` be this element's [URL](#).
3. If `url` or `url`'s [port](#) is null, return the empty string.
4. Return `url`'s [port](#), serialized.

The **port** attribute's setter must run these steps:

1. [Reinitialise url](#).
2. Let *url* be this element's [URL](#).
3. If *url* or *url*'s [host](#) is null, *url*'s [non-relative flag](#) is set, or *url*'s [scheme](#) is "file", terminate these steps.
4. [Basic URL parse](#) the given value, with *url* as *url* and [port state](#) as *state override*.
5. [Update href](#).

The **pathname** attribute's getter must run these steps:

1. [Reinitialise url](#).
2. Let *url* be this element's [URL](#).
3. If *url* is null, return the empty string.
4. If *url*'s [non-relative flag](#) is set, return the first string in *url*'s [path](#).
5. Return "/", followed by the strings in *url*'s [path](#) (including empty strings), separated from each other by "/".

The **pathname** attribute's setter must run these steps:

1. [Reinitialise url](#).
2. Let *url* be this element's [URL](#).
3. If *url* is null or *url*'s [non-relative flag](#) is set, terminate these steps.
4. Set *url*'s [path](#) to the empty list.
5. [Basic URL parse](#) the given value, with *url* as *url* and [path start state](#) as *state override*.
6. [Update href](#).

The **search** attribute's getter must run these steps:

1. [Reinitialise url](#).
2. Let *url* be this element's [URL](#).
3. If *url* is null, or *url*'s [query](#) is either null or the empty string, return the empty string.
4. Return "?", followed by *url*'s [query](#).

The **search** attribute's setter must run these steps:

1. [Reinitialise url](#).
2. Let *url* be this element's [URL](#).
3. If *url* is null, terminate these steps.
4. If the given value is the empty string, set *url*'s [query](#) to null.
5. Otherwise, run these substeps:
 1. Let *input* be the given value with a single leading "?" removed, if any.
 2. Set *url*'s [query](#) to the empty string.
 3. [Basic URL parse](#) *input*, with *url* as *url* and [query state](#) as *state override*, and this element's [node document](#)'s [document's character encoding](#) as *encoding override*.
6. [Update href](#).

The **hash** attribute's getter must run these steps:

1. [Reinitialise url](#).
2. Let *url* be this element's [URL](#).
3. If *url* is null, or *url*'s [fragment](#) is either null or the empty string, return the empty string.
4. Return "#", followed by *url*'s [fragment](#).

The hash attribute's setter must run these steps:

1. [Reinitialise url](#).
2. Let *url* be this element's [URL](#).
3. If *url* is null or *url*'s [scheme](#) is "javascript", terminate these steps.
4. If the given value is the empty string, set *url*'s [fragment](#) to null.
5. Otherwise, run these substeps:
 1. Let *input* be the given value with a single leading "#" removed, if any.
 2. Set *url*'s [fragment](#) to the empty string.
 3. [Basic URL parse](#) *input*, with *url* as *url* and [fragment state](#) as *state override*.
6. [Update href](#).

§ 4.8.4. Following hyperlinks

When a user **follows a hyperlink** created by an element *subject*, optionally with a *hyperlink suffix*, the user agent must run the following steps:

1. Let *replace* be false.
2. Let *source* be the [browsing context](#) that contains the [Document](#) object with which *subject* in question is associated.
3. If the user indicated a specific [browsing context](#) when following the hyperlink, or if the user agent is configured to follow hyperlinks by navigating a particular browsing context, then let *target* be that [browsing context](#). If this is a new [top-level browsing context](#) (e.g., when the user followed the hyperlink using "Open in New Tab"), then *source* must be set as the new [browsing context](#)'s [one permitted sandboxed navigator](#).

Otherwise, if *subject* is an `<a>` or `<area>` element that has a `target` attribute, then let *target* be the [browsing context](#) that is chosen by applying [the rules for choosing a browsing context given a browsing context name](#), using the value of the `target` attribute as the browsing context name. If these rules result in the creation of a new [browsing context](#), set *replace* to true.

Otherwise, if *target* is an `<a>` or `<area>` element with no `target` attribute, but the [Document](#) contains a `<base>` element with a `target` attribute, then let *target* be the [browsing context](#) that is chosen by applying [the rules for choosing a browsing context given a browsing context name](#), using the value of the `target` attribute of the first such `<base>` element as the browsing context name. If these rules result in the creation of a new [browsing context](#), set *replace* to true.

Otherwise, let *target* be the [browsing context](#) that *subject* itself is in.

4. [Parse](#) the [URL](#) given by *subject*'s `href` attribute, relative to *subject*'s [node document](#).
5. If that is successful, let *URL* be the [resulting URL string](#).

Otherwise, if [parsing the URL](#) failed, the user agent may report the error to the user in a user-agent-specific manner, may [queue a task](#) to [navigate](#) the *target* [browsing context](#) to an error page to report the error, or may ignore the error and do nothing. In any case, the user agent must then abort these steps.

6. If there is a *hyperlink suffix*, append it to *URL*.

7. Queue a task to [navigate](#) the *target browsing context* to *URL*. If *replace* is true, the navigation must be performed with [replacement enabled](#). The *source browsing context* must be *source*.

The *task source* for the tasks mentioned above is the [DOM manipulation task source](#).

§ 4.8.5. Downloading resources

In some cases, resources are intended for later use rather than immediate viewing. To indicate that a resource is intended to be downloaded for use later, rather than immediately used, the `download` attribute can be specified on the `<a>` or `<area>` element that creates the [hyperlink](#) to that resource.

The attribute can furthermore be given a value, to specify the file name that user agents are to use when storing the resource in a file system. This value can be overridden by the Content-Disposition HTTP header's filename parameters.

[\[RFC6266\]](#)

In cross-origin situations, the `download` attribute has to be combined with the Content-Disposition HTTP header, specifically with the attachment disposition type, to avoid the user being warned of possibly nefarious activity. (This is to protect users from being made to download sensitive personal or confidential information without their full understanding.)



When a user **downloads a hyperlink** created by an element *subject*, optionally with a *hyperlink suffix*, the user agent must run the following steps:

1. [Parse](#) the *URL* given by *subject*'s `href` attribute, relative to *subject*.
2. If [parsing](#) the *URL* fails, the user agent may report the error to the user in a user-agent-specific manner, may [navigate](#) to an error page to report the error, or may ignore the error and do nothing. In either case, the user agent must abort these steps.
3. Otherwise, let *URL* be the [resulting URL string](#).
4. If there is a *hyperlink suffix*, append it to *URL*.
5. Return to whatever algorithm invoked these steps and continue these steps [in parallel](#).
6. [Fetch](#) *URL* and handle the resulting resource [as a download](#).

When a user agent is to handle a resource obtained from a fetch **as a download**, it should provide the user with a way to save the resource for later use, if a resource is successfully obtained; or otherwise should report any problems downloading the file to the user.

If the user agent needs a file name for a resource being handled [as a download](#), it should select one using the following algorithm.

⚠Warning! This algorithm is intended to mitigate security dangers involved in downloading files from untrusted sites, and user agents are strongly urged to follow it.

1. Let *filename* be the void value.
2. If the resource has a Content-Disposition header, that header specifies the attachment disposition type, and the header includes file name information, then let *filename* have the value specified by the header, and jump to the step labeled *sanitize* below. [\[RFC6266\]](#)
3. Let *interface origin* be the [origin](#) of the [Document](#) in which the [download](#) or [navigate](#) action resulting in the download was initiated, if any.
4. Let *resource origin* be the [origin](#) of the URL of the resource being downloaded, unless that URL's `scheme` component is data, in which case let *resource origin* be the same as the *interface origin*, if any.
5. If there is no *interface origin*, then let *trusted operation* be true. Otherwise, let *trusted operation* be true if *resource*

origin is the [same origin](#) as *interface origin*, and false otherwise.

6. If *trusted operation* is true and the resource has a Content-Disposition header and that header includes file name information, then let *filename* have the value specified by the header, and jump to the step labeled *sanitize* below. [\[RFC6266\]](#)
7. If the download was not initiated from a [hyperlink](#) created by an `<a>` or `<area>` element, or if the element of the [hyperlink](#) from which it was initiated did not have a `download` attribute when the download was initiated, or if there was such an attribute but its value when the download was initiated was the empty string, then jump to the step labeled *no proposed file name*.
8. Let *proposed filename* have the value of the `download` attribute of the element of the [hyperlink](#) that initiated the download at the time the download was initiated.
9. If *trusted operation* is true, let *filename* have the value of *proposed filename*, and jump to the step labeled *sanitize* below.
10. If the resource has a Content-Disposition header and that header specifies the attachment disposition type, let *filename* have the value of *proposed filename*, and jump to the step labeled *sanitize* below. [\[RFC6266\]](#)
11. *No proposed file name*: If *trusted operation* is true, or if the user indicated a preference for having the resource in question downloaded, let *filename* have a value derived from the [URL](#) of the resource in a user-agent-defined manner, and jump to the step labeled *sanitize* below.
12. Act in a user-agent-defined manner to safeguard the user from a potentially hostile cross-origin download. If the download is not to be aborted, then let *filename* be set to the user's preferred file name or to a file name selected by the user agent, and jump to the step labeled *sanitize* below.

If the algorithm reaches this step, then a download was begun from a different origin than the resource being downloaded, and the origin did not mark the file as suitable for downloading, and the download was not initiated by the user. This could be because a `download` attribute was used to trigger the download, or because the resource in question is not of a type that the user agent supports.

This could be dangerous, because, for instance, a hostile server could be trying to get a user to unknowingly download private information and then re-upload it to the hostile server, by tricking the user into thinking the data is from the hostile server.

Thus, it is in the user's interests that the user be somehow notified that the resource in question comes from quite a different source, and to prevent confusion, any suggested file name from the potentially hostile *interface origin* should be ignored.

13. *Sanitize*: Optionally, allow the user to influence *filename*. For example, a user agent could prompt the user for a file name, potentially providing the value of *filename* as determined above as a default value.
14. Adjust *filename* to be suitable for the local file system.

EXAMPLE 439

For example, this could involve removing characters that are not legal in file names, or trimming leading and trailing whitespace.

15. If the platform conventions do not in any way use [extensions](#) to determine the types of file on the file system, then return *filename* as the file name and abort these steps.
16. Let *claimed type* be the type given by the resource's [Content-Type metadata](#), if any is known. Let *named type* be the type given by *filename*'s [extension](#), if any is known. For the purposes of this step, a *type* is a mapping of a [MIME type](#) to an [extension](#).
17. If *named type* is consistent with the user's preferences (e.g., because the value of *filename* was determined by prompting the user), then return *filename* as the file name and abort these steps.
18. If *claimed type* and *named type* are the same type (i.e., the type given by the resource's [Content-Type metadata](#) is con-

sistent with the type given by *filename*'s [extension](#)), then return *filename* as the file name and abort these steps.

19. If the *claimed type* is known, then alter *filename* to add an [extension](#) corresponding to *claimed type*.

Otherwise, if *named type* is known to be potentially dangerous (e.g., it will be treated by the platform conventions as a native executable, shell script, HTML application, or executable-macro-capable document) then optionally alter *filename* to add a known-safe [extension](#) (e.g., ".txt").

NOTE:

This last step would make it impossible to download executables, which might not be desirable. As always, implementors are forced to balance security and usability in this matter.

20. Return *filename* as the file name.

For the purposes of this algorithm, a file **extension** consists of any part of the file name that platform conventions dictate will be used for identifying the type of the file. For example, many operating systems use the part of the file name following the last dot (".") in the file name to determine the type of the file, and from that the manner in which the file is to be opened or executed.

User agents should ignore any directory or path information provided by the resource itself, its [URL](#), and any [download attribute](#), in deciding where to store the resulting file in the user's file system.

§ 4.8.6. Link types

The following table summarizes the link types that are defined by this specification. This table is non-normative; the actual definitions for the link types are given in the next few sections.

In this section, the term *referenced document* refers to the resource identified by the element representing the link, and the term *current document* refers to the resource within which the element representing the link finds itself.

To determine which link types apply to a [<link>](#), [<a>](#), or [<area>](#) element, the element's [rel](#) attribute must be [split on spaces](#). The resulting tokens are the link types that apply to that element.

Except where otherwise specified, a keyword must not be specified more than once per [rel](#) attribute.

Link types are always [ASCII case-insensitive](#), and must be compared as such.

EXAMPLE 440

Thus, `rel="next"` is the same as `rel="NEXT"`.

Link type	Effect on...		Brief description
	link	<a> and area	
alternate	hyperlink	hyperlink	Gives alternate representations of the current document.
author	hyperlink	hyperlink	Gives a link to the author of the current document or article.
bookmark	<i>not allowed</i>	hyperlink	Gives the permalink for the nearest ancestor section.
help	hyperlink	hyperlink	Provides a link to context-sensitive help.
icon	External Resource	<i>not allowed</i>	Imports an icon to represent the current document.
license	hyperlink	hyperlink	Indicates that the main content of the current document is covered by the copyright license described by the referenced document.
next	hyperlink	hyperlink	Indicates that the current document is a part of a series, and that the next document in the series is the referenced document.
nofollow	<i>not allowed</i>	Annotation	Indicates that the current document's original author or publisher does not endorse the referenced document.
noreferrer	<i>not allowed</i>	Annotation	Requires that the user agent not send an HTTP Referer (sic) header if the user follows the hyperlink.

Link type	Effect on...		Brief description
	link	<a> and area	
prev	hyperlink	hyperlink	Indicates that the current document is a part of a series, and that the previous document in the series is the referenced document.
search	hyperlink	hyperlink	Gives a link to a resource that can be used to search through the current document and its related pages.
stylesheet	External Resource	<i>not allowed</i>	Imports a stylesheet.
tag	<i>not allowed</i>	hyperlink	Gives a tag (identified by the given address) that applies to the current document.

Some of the types described below list synonyms for these values. These are to be handled as specified by user agents, but must not be used in documents.

4.8.6.1. Link type "alternate"

The alternate keyword may be used with [<link>](#), [<a>](#), and [<area>](#) elements.

The meaning of this keyword depends on the values of the other attributes.

↪ If the element is a [<link>](#) element and the [rel](#) attribute also contains the keyword **stylesheet**

The alternate keyword modifies the meaning of the stylesheet keyword in the way described for that keyword. The alternate keyword does not create a link of its own.

↪ If the alternate keyword is used with the type attribute set to the value **application/rss+xml** or the value **application/atom+xml**

The keyword creates a [hyperlink](#) referencing a syndication feed (though not necessarily syndicating exactly the same content as the current page).

The first link or [<a>](#) element in the document (in [tree order](#)) with the alternate keyword used with the type attribute set to the value **application/rss+xml** or the value **application/atom+xml** must be treated as the default syndication feed for the purposes of feed autodiscovery.

EXAMPLE 441

The following [<link>](#) element gives the syndication feed for the current page:

```
<link rel="alternate" type="application/atom+xml" href="data.xml">
```

The following extract offers various different syndication feeds:

```
<p>You can access the planets database using Atom feeds:</p>
<ul>
  <li><a href="recently-visited-planets.xml" rel="alternate"
type="application/atom+xml">Recently Visited Planets</a></li>
  <li><a href="known-bad-planets.xml" rel="alternate"
type="application/atom+xml">Known Bad Planets</a></li>
  <li><a href="unexplored-planets.xml" rel="alternate"
type="application/atom+xml">Unexplored Planets</a></li>
</ul>
```

↪ Otherwise

The keyword creates a [hyperlink](#) referencing an alternate representation of the current document.

The nature of the referenced document is given by the [hreflang](#), and [type](#) attributes.

If the alternate keyword is used with the [hreflang](#) attribute, and that attribute's value differs from the [root element's language](#), it indicates that the referenced document is a translation.

If the `alternate` keyword is used with the `type` attribute, it indicates that the referenced document is a reformulation of the current document in the specified format.

The `hreflang` and `type` attributes can be combined when specified with the `alternate` keyword.

EXAMPLE 442

For example, the following link is a French translation that uses the PDF format:

```
<link rel=alternate type=application/pdf hreflang=fr href>manual-fr>
```

This relationship is transitive — that is, if a document links to two other documents with the link type "alternate", then, in addition to implying that those documents are alternative representations of the first document, it is also implying that those two documents are alternative representations of each other.

§ 4.8.6.2. Link type "author"

The `author` keyword may be used with `<link>`, `<a>`, and `<area>` elements. This keyword creates a [hyperlink](#).

For `<a>` and `<area>` elements, the `author` keyword indicates that the referenced document provides further information about the author of the nearest `<article>` element ancestor of the element defining the hyperlink, if there is one, or of the page as a whole, otherwise.

For `<link>` elements, the `author` keyword indicates that the referenced document provides further information about the author for the page as a whole.

NOTE:

The "referenced document" can be, and often is, a `mailto:` URL giving the e-mail address of the author. [\[RFC6068\]](#)

Synonyms: For historical reasons, user agents must also treat `<link>`, `<a>`, and `<area>` elements that have a `rev` attribute with the value "made" as having the `author` keyword specified as a link relationship.

§ 4.8.6.3. Link type "bookmark"

The `bookmark` keyword may be used with `<a>` and `<area>` elements. This keyword creates a [hyperlink](#).

The `bookmark` keyword gives a permalink for the nearest ancestor `<article>` element of the linking element in question, or of [the section the linking element is most closely associated with](#), if there are no ancestor `<article>` elements.

EXAMPLE 443

The following snippet has three permalinks. A user agent could determine which permalink applies to which part of the spec by looking at where the permalinks are given.

```
...
<body>
<h1>Example of permalinks</h1>
<div id="a">
  <h2>First example</h2>
  <p><a href="a.html" rel="bookmark">This permalink applies to
    only the content from the first H2 to the second H2</a>. The DIV isn't
    exactly that section, but it roughly corresponds to it.</p>
</div>
<h2>Second example</h2>
<article id="b">
  <p><a href="b.html" rel="bookmark">This permalink applies to
    the outer ARTICLE element</a> (which could be, e.g., a blog post).</p>
  <article id="c">
    <p><a href="c.html" rel="bookmark">This permalink applies to
      the inner ARTICLE element</a> (which could be, e.g., a blog comment).</p>
  </article>
</article>
</body>
...
```

4.8.6.4. Link type "help"

The `help` keyword may be used with `<link>`, `<a>`, and `<area>` elements. This keyword creates a [hyperlink](#).

For `<a>` and `<area>` elements, the `help` keyword indicates that the referenced document provides further help information for the parent of the element defining the hyperlink, and its children.

EXAMPLE 444

In the following example, the form control has associated context-sensitive help. The user agent could use this information, for example, displaying the referenced document if the user presses the "Help" or "F1" key.

```
<p><label> Topic: <input name=topic> <a href="help/topic.html" rel="help">(Help)
</a></label></p>
```

For `<link>` elements, the `help` keyword indicates that the referenced document provides help for the page as a whole.

For `<a>` and `<area>` elements, on some browsers, the `help` keyword causes the link to use a different cursor.

4.8.6.5. Link type "icon"

The `icon` keyword may be used with `<link>` elements. This keyword creates an [external resource link](#).

The specified resource is an icon representing the page or site, and should be used by the user agent when representing the page in the user interface.

Icons could be auditory icons, visual icons, or other kinds of icons. If multiple icons are provided, the user agent must select the most appropriate icon according to the `type`, `media`, and `sizes` attributes. If there are multiple equally appropriate icons, user agents must use the last one declared in [tree order](#) at the time that the user agent collected the list of icons. If the user agent tries to use an icon but that icon is determined, upon closer examination, to in fact be inappropriate (e.g., because it uses an unsupported format), then the user agent must try the next-most-appropriate icon as determined by the attributes.

NOTE:

User agents are not required to update icons when the list of icons changes, but are encouraged to do so.

There is no default type for resources given by the `icon` keyword. However, for the purposes of [determining the type of the resource](#), user agents must expect the resource to be an image.

The **sizes** attribute gives the sizes of icons for visual media. Its value, if present, is merely advisory. User agents may use the value to decide which icon(s) to use if multiple icons are available.

If specified, the attribute must have a value that is an [unordered set of unique space-separated tokens](#) which are [ASCII case-insensitive](#). Each value must be either an [ASCII case-insensitive](#) match for the string "any", or a value that consists of two [valid non-negative integers](#) that do not have a leading U+0030 DIGIT ZERO (0) character and that are separated by a single U+0078 LATIN SMALL LETTER X or U+0058 LATIN CAPITAL LETTER X character.

The keywords represent icon sizes in raw pixels (as opposed to CSS pixels).

NOTE:

An icon that is 50 CSS pixels wide intended for displays with a device pixel density of two device pixels per CSS pixel (2x, 192dpi) would have a width of 100 raw pixels. This feature does not support indicating that a different resource is to be used for small high-resolution icons vs large low-resolution icons (e.g., 50×50 2x vs 100×100 1x).

To parse and process the attribute's value, the user agent must first [split the attribute's value on spaces](#), and must then parse each resulting keyword to determine what it represents.

The **any** keyword represents that the resource contains a scalable icon, e.g., as provided by an SVG image.

Other keywords must be further parsed as follows to determine what they represent:

- If the keyword doesn't contain exactly one U+0078 LATIN SMALL LETTER X or U+0058 LATIN CAPITAL LETTER X character, then this keyword doesn't represent anything. Abort these steps for that keyword.
- Let *width string* be the string before the "x" or "X".
- Let *height string* be the string after the "x" or "X".
- If either *width string* or *height string* start with a U+0030 DIGIT ZERO (0) character or contain any characters other than [ASCII digits](#), then this keyword doesn't represent anything. Abort these steps for that keyword.
- Apply the [rules for parsing non-negative integers](#) to *width string* to obtain *width*.
- Apply the [rules for parsing non-negative integers](#) to *height string* to obtain *height*.
- The keyword represents that the resource contains a bitmap icon with a width of *width* device pixels and a height of *height* device pixels.

The keywords specified on the `sizes` attribute must not represent icon sizes that are not actually available in the linked resource.

In the absence of a link with the `icon` keyword, for [Document](#) objects obtained over HTTP or HTTPS, user agents may instead run these steps [in parallel](#):

1. Let *request* be a new [request](#) whose [URL](#) is the [absolute URL](#) obtained by resolving the [URL](#) `"/favicon.ico"` against [the document's address](#), [client](#) is the [Document](#) object's Window object's [environment settings object](#), [type](#) is "image", [destination](#) is "subresource", [synchronous flag](#) is set, [credentials mode](#) is "include", and whose [use-URL-credentials flag](#) is set.
2. Let *response* be the result of [fetching](#) *request*.
3. Use *response*'s unsafe response as an icon as if it had been declared using the `icon` keyword.

EXAMPLE 445

The following snippet shows the top part of an application with several icons.

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>lsForums – Inbox</title>
    <link rel=icon href=favicon.png sizes="16x16" type="image/png">
    <link rel=icon href=windows.ico sizes="32x32 48x48" type="image/vnd.microsoft.icon">
    <link rel=icon href=mac.icns sizes="128x128 512x512 8192x8192 32768x32768">
    <link rel=icon href=iphone.png sizes="57x57" type="image/png">
    <link rel=icon href=gnome.svg sizes="any" type="image/svg+xml">
    <link rel=stylesheet href=lsforums.css>
    <script src=lsforums.js></script>
    <meta name=application-name content="lsForums">
  </head>
  <body>
    ...
```

For historical reasons, the `icon` keyword may be preceded by the keyword `"shortcut"`. If the `"shortcut"` keyword is present, the `rel` attribute's entire value must be an [ASCII case-insensitive](#) match for the string `"shortcut icon"` (with a single U+0020 SPACE character between the tokens and no other [space characters](#)).

4.8.6.6. Link type "License"

The `license` keyword may be used with [<link>](#), [<a>](#), and [<area>](#) elements. This keyword creates a [hyperlink](#).

The `license` keyword indicates that the referenced document provides the copyright license terms under which the main content of the current document is provided.

This specification defines the main content of a document and content that is not deemed to be part of that main content via the [<main>](#) element. The distinction should be made clear to the user.

EXAMPLE 446

Consider a photo sharing site. A page on that site might describe and show a photograph, and the page might be marked up as follows:

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Exapl Pictures: Kissat</title>
    <link rel="stylesheet" href="/style/default">
  </head>
  <body>
    <h1>Kissat</h1>
    <nav>
      <a href="..">Return to photo index</a>
    </nav>

    <main>
      <figure>
        
        <figcaption>Kissat</figcaption>
      </figure>
      <p>One of them has six toes!</p>
      <p><small>This photograph is <a rel="license" href="https://www.opensource.org/licenses/mit-license.php">MIT Licensed</a></small></p>
    </main>
    <footer>
      <a href="/">Home</a> | <a href="..">Photo index</a>
      <p><small>© copyright 2009 Exapl Pictures. All Rights Reserved.</small></p>
    </footer>
  </body>
</html>
```

In this case the `license` applies to just the photo (the main content of the document), not the whole document. In particular not the design of the page itself, which is covered by the copyright given at the bottom of the document. This should be made clear in the text referencing the licensing link and could also be made clearer in the styling (e.g., making the license link prominently positioned near the photograph, while having the page copyright in small text at the foot of the page, or adding a border to the `<main>` element.)

Synonyms: For historical reasons, user agents must also treat the keyword `"copyright"` like the `license` keyword.

4.8.6.7. Link type `"noFollow"`

The `noFollow` keyword may be used with `<a>` and `<area>` elements. This keyword does not create a [hyperlink](#), but [annotates](#) any other hyperlinks created by the element (the implied hyperlink, if no other keywords create one).

The `noFollow` keyword indicates that the link is not endorsed by the original author or publisher of the page, or that the link to the referenced document was included primarily because of a commercial relationship between people affiliated with the two pages.

4.8.6.8. Link type `"noReferrer"`

The `noReferrer` keyword may be used with `<a>` and `<area>` elements. This keyword does not create a [hyperlink](#), but [annotates](#) any other hyperlinks created by the element (the implied hyperlink, if no other keywords create one).

It indicates that no referrer information is to be leaked when following the link.

If a user agent follows a link defined by an `<a>` or `<area>` element that has the `noReferrer` keyword, the user agent must set their [request's referrer](#) to `"no-referrer"`.

This keyword also [causes the opener attribute to remain null](#) if the hyperlink creates a new [browsing context](#).

4.8.6.9. Link type "search"

The search keyword may be used with `<link>`, `<a>`, and `<area>` elements. This keyword creates a [hyperlink](#).

The search keyword indicates that the referenced document provides an interface specifically for searching the document and its related resources.

NOTE:

OpenSearch description documents can be used with `<link>` elements and the search link type to enable user agents to autodiscover search interfaces. [\[OPENSEARCH\]](#)

4.8.6.10. Link type "stylesheet"

The stylesheet keyword may be used with `link` elements. This keyword creates an [external resource link](#) that contributes to the styling processing model.

The specified resource is a resource that describes how to present the document. Exactly how the resource is to be processed depends on the actual type of the resource.

If the alternate keyword is also specified on the `<link>` element, then **the link is an alternative stylesheet**; in this case, the title attribute must be specified on the `link` element, with a non-empty value.

The default type for resources given by the stylesheet keyword is `text/css`.

The appropriate times to [obtain](#) the resource are:

- When the [external resource link](#) is created on a `<link>` element that is already [in a Document](#).
- When the [external resource link](#)'s `<link>` element is [inserted into a document](#).
- When the `href` attribute of the `<link>` element of an [external resource link](#) that is already [in a Document](#) is changed.
- When the `crossorigin` attribute of the `<link>` element of an [external resource link](#) that is already [in a Document](#) is set, changed, or removed.
- When the `type` attribute of the `<link>` element of an [external resource link](#) that is already [in a Document](#) is set or changed to a value that does not or no longer matches the [Content-Type metadata](#) of the previous obtained external resource, if any.
- When the `type` attribute of the `<link>` element of an [external resource link](#) that is already [in a Document](#) but was previously not obtained due to the type attribute specifying an unsupported type is set, removed, or changed.
- When the [external resource link](#) changes from being [an alternative stylesheet](#) to not being one, or vice versa.

Quirk: If the document has been set to [quirks mode](#), has the [same origin](#) as the [URL](#) of the external resource, and the [Content-Type metadata](#) of the external resource is not a supported style sheet type, the user agent must instead assume it to be `text/css`.

Once a resource has been [obtained](#), if its [Content-Type metadata](#) is `text/css`, the user agent must run these steps:

1. Let *element* be the `<link>` element that created the [external resource link](#).
2. If *element* has an [associated CSS style sheet](#), [remove the CSS style sheet](#) in question.
3. If *element* no longer creates an [external resource link](#) that contributes to the styling processing model, or if, since the resource in question was [obtained](#), it has become appropriate to [obtain](#) it again (meaning this algorithm is about to be invoked again for a newly obtained resource), then abort these steps.
4. [Create a CSS style sheet](#) with the following properties:

[type](#)

`text/css`

[location](#)

The [resulting URL string](#) determined during the [obtain](#) algorithm.

NOTE:

This is before any redirects get applied.

owner node

element

media

The `media` attribute of *element*.

NOTE:

This is a reference to the (possibly absent at this time) attribute, rather than a copy of the attribute's current value. The CSSOM specification defines what happens when the attribute is dynamically set, changed, or removed.

title

The `title` attribute of *element*.

NOTE:

This is similarly a reference to the attribute, rather than a copy of the attribute's current value.

alternate flag

Set if [the link is an alternative stylesheet](#); unset otherwise.

origin-clean flag

Set if the resource is [CORS-same-origin](#); unset otherwise.

parent CSS style sheet**owner CSS rule**

null

disabled flag

Left at its default value.

CSS rules

Left uninitialized.

The CSS [environment encoding](#) is the result of running the following steps: [\[CSS-SYNTAX-3\]](#)

1. If the element has a `charset` attribute, [get an encoding](#) from that attribute's value. If that succeeds, return the resulting encoding and abort these steps. [\[ENCODING\]](#)
2. Otherwise, return the [document's character encoding](#). [\[DOM\]](#)

4.8.6.11. Link type "tag"

The `tag` keyword may be used with `<a>` and `<area>` elements. This keyword creates a [hyperlink](#).

The `tag` keyword indicates that the *tag* that the referenced document represents applies to the current document.

NOTE:

Since it indicates that the tag applies to the current document, it would be inappropriate to use this keyword in the markup of a [tag cloud](#), which lists the popular tags across a set of pages.

EXAMPLE 447

This document is about some gems, and so it is *tagged* with "https://en.wikipedia.org/wiki/Gemstone" to unambiguously categorize it as applying to the "jewel" kind of gems, and not to, say, the towns in the US, the Ruby package format, or the Swiss locomotive class:

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>My Precious</title>
  </head>
  <body>
    <header><h1>My precious</h1> <p>Summer 2012</p></header>
    <p>Recently I managed to dispose of a red gem that had been
    bothering me. I now have a much nicer blue sapphire.</p>
    <p>The red gem had been found in a bauxite stone while I was digging
    out the office level, but nobody was willing to haul it away. The
    same red gem stayed there for literally years.</p>
  <footer>
    Tags: <a rel=tag href="https://en.wikipedia.org/wiki/Gemstone">Gemstone</a>
  </footer>
</body>
</html>
```

EXAMPLE 448

In *this* document, there are two articles. The "tag" link, however, applies to the whole page (and would do so wherever it was placed, including if it was within the `<article>` elements).

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Gem 4/4</title>
  </head>
  <body>
    <article>
      <h1>801: Steinbock</h1>
      <p>The number 801 Gem 4/4 electro-diesel has an ibex and was rebuilt in 2002.</p>
    </article>
    <article>
      <h1>802: Murmeltier</h1>
      <figure>
        
        <figcaption>The 802 in the 1980s, above Lago Bianco.</figcaption>
      </figure>
      <p>The number 802 Gem 4/4 electro-diesel has a marmot and was rebuilt in 2003.</p>
    </article>
    <p class="topic"><a rel=tag href="https://en.wikipedia.org/wiki/Rhaetian_Railway_Gem_4
    /4">Gem 4/4</a></p>
  </body>
</html>
```

4.8.6.12. Sequential link types

Some documents form part of a sequence of documents.

A sequence of documents is one where each document can have a *previous sibling* and a *next sibling*. A document with no previous sibling is the start of its sequence, a document with no next sibling is the end of its sequence.

A document may be part of multiple sequences.

§ 4.8.6.12.1. LINK TYPE "next"

The next keyword may be used with `<link>`, `<a>`, and `<area>` elements. This keyword creates a [hyperlink](#).

The next keyword indicates that the document is part of a sequence, and that the link is leading to the document that is the next logical document in the sequence.

§ 4.8.6.12.2. LINK TYPE "prev"

The prev keyword may be used with `<link>`, `<a>`, and `<area>` elements. This keyword creates a [hyperlink](#).

The prev keyword indicates that the document is part of a sequence, and that the link is leading to the document that is the previous logical document in the sequence.

Synonyms: For historical reasons, user agents must also treat the keyword "previous" like the prev keyword.

§ 4.8.6.13. Other link types

Extensions to the predefined set of link types may be registered in the [microformats wiki existing-rel-values page](#). [\[MFREL\]](#)

Anyone is free to edit the [microformats wiki existing-rel-values page](#) at any time to add a type. Extension types must be specified with the following information:

Keyword

The actual value being defined. The value should not be confusingly similar to any other defined value (e.g., differing only in case).

If the value contains a U+003A COLON character (:), it must also be an [absolute URL](#).

Effect on... link

One of the following:

Not allowed

The keyword must not be specified on `<link>` elements.

Hyperlink

The keyword may be specified on a `<link>` element; it creates a [hyperlink](#).

External Resource

The keyword may be specified on a `<link>` element; it creates an [external resource link](#).

Effect on... `<a>` and `area`

One of the following:

Not allowed

The keyword must not be specified on `<a>` and `<area>` elements.

Hyperlink

The keyword may be specified on `<a>` and `<area>` elements; it creates a [hyperlink](#).

External Resource

The keyword may be specified on `<a>` and `<area>` elements; it creates an [external resource link](#).

Hyperlink Annotation

The keyword may be specified on `<a>` and `<area>` elements; it [annotates](#) other [hyperlinks](#) created by the element.

Brief description

A short non-normative description of what the keyword's meaning is.

Specification

A link to a more detailed description of the keyword's semantics and requirements. It could be another page on the Wiki, or a link to an external page.

Synonyms

A list of other keyword values that have exactly the same processing requirements. Authors should not use the values defined to be synonyms, they are only intended to allow user agents to support legacy content. Anyone may remove synonyms that are not used in practice; only names that need to be processed as synonyms for compatibility with legacy content are to be registered in this way.

Status

One of the following:

Proposed

The keyword has not received wide peer review and approval. Someone has proposed it and is, or soon will be, using it.

Ratified

The keyword has received wide peer review and approval. It has a specification that unambiguously defines how to handle pages that use the keyword, including when they use it in incorrect ways.

Discontinued

The keyword has received wide peer review and it has been found wanting. Existing pages are using this keyword, but new pages should avoid it. The "brief description" and "specification" entries will give details of what authors should use instead, if anything.

If a keyword is found to be redundant with existing values, it should be removed and listed as a synonym for the existing value.

If a keyword is registered in the "proposed" state for a period of a month or more without being used or specified, then it may be removed from the registry.

If a keyword is added with the "proposed" status and found to be redundant with existing values, it should be removed and listed as a synonym for the existing value. If a keyword is added with the "proposed" status and found to be harmful, then it should be changed to "discontinued" status.

Anyone can change the status at any time, but should only do so in accordance with the definitions above.

Conformance checkers may use the information given on the [microformats wiki existing-rel-values page](#) to establish if a value is allowed or not: values defined in this specification or marked as "proposed" or "ratified" must be accepted when used on the elements for which they apply as described in the "Effect on..." field, whereas values marked as "discontinued" or values not containing a U+003A COLON character but not listed in either this specification or on the aforementioned page must be reported as invalid. The remaining values must be accepted as valid if they are absolute URLs containing US-ASCII characters only and rejected otherwise. Conformance checkers may cache this information (e.g., for performance reasons or to avoid the use of unreliable network connectivity).

NOTE:

Note: Even URL-valued link types are compared ASCII-case-insensitively. Validators might choose to warn about characters U+0041 (LATIN CAPITAL LETTER A) through U+005A (LATIN CAPITAL LETTER Z) (inclusive) in the pre-case-folded form of link types that contain a colon.

When an author uses a new type not defined by either this specification or the Wiki page, conformance checkers should offer to add the value to the Wiki, with the details described above, with the "proposed" status.

Types defined as extensions in the [microformats wiki existing-rel-values page](#) with the status "proposed" or "ratified" may be used with the [rel](#) attribute on [<link>](#), [<a>](#), and [<area>](#) elements in accordance to the "Effect on..." field. [\[MFREL\]](#)

4.9. Tabular data

4.9.1. The table element

Categories:

[Flow content.](#)

[Palpable content.](#)

Contexts in which this element can be used:

Where [flow content](#) is expected.

Content model:

In this order: optionally a `<caption>` element, followed by zero or more `<colgroup>` elements, followed optionally by a `<thead>` element, followed by either zero or more `<tbody>` elements or one or more `<tr>` elements, followed optionally by a `<tfoot>` element, optionally intermixed with one or more [script-supporting elements](#).

Tag omission in text/html:

Neither tag is omissible

Content attributes:

Global attributes

`border`

Allowed ARIA role attribute values:

[Any role value.](#)

Allowed ARIA state and property attributes:

Global aria-* attributes

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLTableElement : HTMLElement {
  attribute HTMLTableCaptionElement? caption;
  HTMLTableCaptionElement createCaption();
  void deleteCaption();
  attribute HTMLTableSectionElement? tHead;
  HTMLTableSectionElement createTHead();
  void deleteTHead();
  attribute HTMLTableSectionElement? tFoot;
  HTMLTableSectionElement createTFoot();
  void deleteTFoot();
  [SameObject] readonly attribute HTMLCollection tBodies;
  HTMLTableSectionElement createTBody();
  [SameObject] readonly attribute HTMLCollection rows;
  HTMLTableRowElement insertRow(optional long index = -1);
  void deleteRow(long index);
};
```

The `<table>` element [represents](#) data with more than one dimension, in the form of a [table](#).

The `<table>` element takes part in the [table model](#). Tables have rows, columns, and cells given by their descendants. The rows and columns form a grid; a table's cells must completely cover that grid without overlap.

NOTE:

Precise rules for determining whether this conformance requirement is met are described in the description of the [table model](#).

Authors are encouraged to provide information describing how to interpret complex tables. Guidance on how to [provide such information](#) is given below.

Tables should not be used as layout aids.

Historically, many Web authors have tables in HTML as a way to control their page layout making it difficult to extract tabular data from such documents.

In particular, users of accessibility tools, like screen readers, are likely to find it very difficult to navigate pages with tables used for layout.

If a table is to be used for layout it must be marked with the attribute `role="presentation"` for a user agent to properly represent the table to an assistive technology and to properly convey the intent of the author to tools that wish to extract tabular data from the document.

NOTE:

There are a variety of alternatives to using HTML tables for layout, primarily using CSS positioning and the CSS table model. [\[CSS-2015\]](#)

The **border** content attribute may be specified on a `<table>` element to explicitly indicate that the `<table>` element is not being used for layout purposes. If specified, the attribute's value must either be the empty string or the value "1". The attribute is used by certain user agents as an indication that borders should be drawn around cells of the table.



Tables can be complicated to understand and navigate. To help users with this, user agents should clearly delineate cells in a table from each other, unless the user agent has classified the table as a layout table.

NOTE:

Authors and implementors are encouraged to consider using some of the [table design techniques](#) described below to make tables easier to navigate for users.

User agents, especially those that do table analysis on arbitrary content, are encouraged to find heuristics to determine which tables actually contain data and which are merely being used for layout. This specification does not define a precise heuristic, but the following are suggested as possible indicators:

Feature	Indication
The use of the <code>role</code> attribute with the value <code>presentation</code>	Probably a layout table
The use of the non-conforming <code>border</code> attribute with the non-conforming value 0	Probably a layout table
The use of the non-conforming <code>cellspacing</code> and <code>cellpadding</code> attributes with the value 0	Probably a layout table
The use of <code><caption></code> , <code><thead></code> , or <code><th></code> elements	Probably a non-layout table
The use of the <code>headers</code> and <code>scope</code> attributes	Probably a non-layout table
The use of the non-conforming <code>border</code> attribute with a value other than 0	Probably a non-layout table
Explicit visible borders set using CSS	Probably a non-layout table
The use of the <code>summary</code> attribute	Not a good indicator (both layout and non-layout tables have historically been given this attribute)

NOTE:

It is quite possible that the above suggestions are wrong. Implementors are urged to provide feedback elaborating on their experiences with trying to create a layout table detection heuristic.

If a `<table>` element has a (non-conforming) `summary` attribute, and the user agent has not classified the table as a layout table, the user agent may report the contents of that attribute to the user.



This definition is non-normative. Implementation requirements are given below this definition.

`table . caption [ = value ]`

Returns the table's `<caption>` element.

Can be set, to replace the `<caption>` element.

`caption = table . createCaption()`

Ensures the table has a `<caption>` element, and returns it.

`table . deleteCaption()`

Ensures the table does not have a [<caption>](#) element.

`table . tHead [ = value ]`

Returns the table's [<thead>](#) element.

Can be set, to replace the [<thead>](#) element. If the new value is not a [<thead>](#) element, throws a `HierarchyRequestError` exception.

`thead = table . createTHead()`

Ensures the table has a [<thead>](#) element, and returns it.

`table . deleteTHead()`

Ensures the table does not have a [<thead>](#) element.

`table . tFoot [ = value ]`

Returns the table's [<tfoot>](#) element.

Can be set, to replace the [<tfoot>](#) element. If the new value is not a [<tfoot>](#) element, throws a `HierarchyRequestError` exception.

`tfoot = table . createTFoot()`

⚠Warning! There is only one known native implementation of `createTFoot` (Firefox/Gecko). Therefore this feature should not be relied upon.

Ensures the table has a [<tfoot>](#) element, and returns it.

`table . deleteTFoot()`

Ensures the table does not have a [<tfoot>](#) element.

`table . tBodies`

Returns an `HTMLCollection` of the [<tbody>](#) elements of the table.

`tbody = table . createTBody()`

Creates a [<tbody>](#) element, inserts it into the table, and returns it.

`table . rows`

Returns an `HTMLCollection` of the [<tr>](#) elements of the table.

`tr = table . insertRow( [ index ] )`

Creates a [<tr>](#) element, along with a `tbody` if required, inserts them into the table at the position given by the argument, and returns the [<tr>](#).

The position is relative to the rows in the table. The index -1, which is the default if the argument is omitted, is equivalent to inserting at the end of the table.

If the given position is less than -1 or greater than the number of rows, throws an `IndexSizeError` exception.

`table . deleteRow(index)`

Removes the [<tr>](#) element with the given position in the table.

The position is relative to the rows in the table. The index -1 is equivalent to deleting the last row of the table.

If the given position is less than -1 or greater than the index of the last row, or if there are no rows, throws an `IndexSizeError` exception.

The **`caption`** IDL attribute must return, on getting, the first [<caption>](#) element child of the [<table>](#) element, if any, or null otherwise. On setting, the first [<caption>](#) element child of the [<table>](#) element, if any, must be removed, and the new value, if not null, must be inserted as the first node of the [<table>](#) element.

The **`createCaption()`** method must return the first [<caption>](#) element child of the [<table>](#) element, if any; otherwise a new [<caption>](#) element must be created, inserted as the first node of the [<table>](#) element, and then returned.

The **`deleteCaption()`** method must remove the first [<caption>](#) element child of the [<table>](#) element, if any.

The **tHead** IDL attribute must return, on getting, the first `<thead>` element child of the `<table>` element, if any, or null otherwise. On setting, if the new value is null or a `<thead>` element, the first `<thead>` element child of the `<table>` element, if any, must be removed, and the new value, if not null, must be inserted immediately before the first element in the `<table>` element that is neither a `<caption>` element nor a `<colgroup>` element, if any, or at the end of the table if there are no such elements. If the new value is neither null nor a `<thead>` element, then a `HierarchyRequestError` DOM exception must be thrown instead.

The **createTHead()** method must return the first `<thead>` element child of the `<table>` element, if any; otherwise a new `<thead>` element must be created and inserted immediately before the first element in the `<table>` element that is neither a `<caption>` element nor a `<colgroup>` element, if any, or at the end of the table if there are no such elements, and then that new element must be returned.

The **deleteTHead()** method must remove the first `<thead>` element child of the `<table>` element, if any.

The **tFoot** IDL attribute must return, on getting, the first `<tfoot>` element child of the `<table>` element, if any, or null otherwise. On setting, if the new value is null or a `<tfoot>` element, the first `<tfoot>` element child of the `<table>` element, if any, must be removed, and the new value, if not null, must be inserted at the end of the table. If the new value is neither null nor a `<tfoot>` element, then a `HierarchyRequestError` DOM exception must be thrown instead.

The **createTFoot()** method must return the first `<tfoot>` element child of the `<table>` element, if any; otherwise a new `<tfoot>` element must be created and inserted at the end of the table, and then that new element must be returned.

The **deleteTFoot()** method must remove the first `<tfoot>` element child of the `<table>` element, if any.

The **tBodies** attribute must return an `HTMLCollection` rooted at the `table` node, whose filter matches only `<tbody>` elements that are children of the `<table>` element.

The **createTBody()** method must create a new `<tbody>` element, insert it immediately after the last `<tbody>` element child in the `<table>` element, if any, or at the end of the `<table>` element if the `<table>` element has no `<tbody>` element children, and then must return the new `<tbody>` element.

The **rows** attribute must return an `HTMLCollection` rooted at the `table` node, whose filter matches only `<tr>` elements that are either children of the `<table>` element, or children of `<thead>`, `<tbody>`, or `<tfoot>` elements that are themselves children of the `<table>` element. The elements in the collection must be ordered such that those elements whose parent is a `thead` are included first, in [tree order](#), followed by those elements whose parent is either a `table` or `tbody` element, again in [tree order](#), followed finally by those elements whose parent is a `<tfoot>` element, still in [tree order](#).

The behavior of the **insertRow(*index*)** method depends on the state of the table. When it is called, the method must act as required by the first item in the following list of conditions that describes the state of the table and the *index* argument:

↪ **If *index* is less than -1 or greater than the number of elements in rows collection:**

The method must throw an `IndexSizeError` exception.

↪ **If the rows collection has zero elements in it, and the table has no `<tbody>` elements in it:**

The method must create a `<tbody>` element, then create a `<tr>` element, then append the `<tr>` element to the `<tbody>` element, then append the `<tbody>` element to the `<table>` element, and finally return the `<tr>` element.

↪ **If the rows collection has zero elements in it:**

The method must create a `<tr>` element, append it to the last `tbody` element in the table, and return the `<tr>` element.

↪ **If *index* is -1 or equal to the number of items in rows collection:**

The method must create a `<tr>` element, and append it to the parent of the last `<tr>` element in the `rows` collection. Then, the newly created `<tr>` element must be returned.

↪ **Otherwise:**

The method must create a `<tr>` element, insert it immediately before the *index*th `<tr>` element in the `rows` collection, in the same parent, and finally must return the newly created `tr` element.

When the **deleteRow(*index*)** method is called, the user agent must run the following steps:

1. If *index* is equal to -1, then *index* must be set to the number of items in the `rows` collection, minus one.

2. Now, if *index* is less than zero, or greater than or equal to the number of elements in the *rows* collection, the method must instead throw an `IndexSizeError` exception, and these steps must be aborted.
3. Otherwise, the method must remove the *index*th element in the *rows* collection from its parent.

EXAMPLE 449

Here is an example of a table being used to mark up a Sudoku puzzle. Observe the lack of headers, which are not necessary in such a table.

```
<section>
  <h1>Today's Sudoku</h1>
  <table>
    <colgroup><col><col><col>
    <colgroup><col><col><col>
    <colgroup><col><col><col>
    <tbody>
      <tr> <td> 1 <td>   <td> 3 <td> 6 <td>   <td> 4 <td> 7 <td>   <td> 9
      <tr> <td>   <td> 2 <td>   <td>   <td> 9 <td>   <td>   <td> 1 <td>
      <tr> <td> 7 <td>   <td>   <td>   <td>   <td>   <td>   <td> 6
    </tbody>
    <tbody>
      <tr> <td> 2 <td>   <td> 4 <td>   <td> 3 <td>   <td> 9 <td>   <td> 8
      <tr> <td>   <td>   <td>   <td>   <td>   <td>   <td>   <td>   <td>
      <tr> <td> 5 <td>   <td>   <td> 9 <td>   <td> 7 <td>   <td>   <td> 1
    </tbody>
    <tbody>
      <tr> <td> 6 <td>   <td>   <td>   <td> 5 <td>   <td>   <td> 2
      <tr> <td>   <td>   <td>   <td>   <td> 7 <td>   <td>   <td>   <td>
      <tr> <td> 9 <td>   <td>   <td> 8 <td>   <td> 2 <td>   <td>   <td> 5
    </tbody>
  </table>
</section>
```

4.9.1.1. Techniques for describing tables

For tables that consist of more than just a grid of cells with headers in the first row and headers in the first column, and for any table in general where the reader might have difficulty understanding the content, authors should include explanatory information introducing the table. This information is useful for all users, but is especially useful for users who cannot see the table, e.g., users of screen readers.

Such explanatory information should introduce the purpose of the table, outline its basic cell structure, highlight any trends or patterns, and generally teach the user how to use the table.

For instance, the following table:

Characteristics with positive and negative sides

Negative	Characteristic	Positive
Sad	Mood	Happy
Failing	Grade	Passing

...could benefit from a description explaining the way the table is laid out, something like "Characteristics are given in the second column, with the negative side in the left column and the positive side in the right column".

There are a variety of ways to include this information, such as:

In prose, surrounding the table

EXAMPLE 450

```

<p id="summary">In the following table, characteristics are
given in the second column, with the negative side in the left column and the positive
side in the right column.</p>
<table aria-describedby="summary">
  <caption>Characteristics with positive and negative sides</caption>
  <thead>
    <tr>
      <th id="n"> Negative
      <th> Characteristic
      <th> Positive
    </tr>
  </thead>
  <tbody>
    <tr>
      <td headers="n r1"> Sad
      <th id="r1"> Mood
      <td> Happy
    </tr>
    <tr>
      <td headers="n r2"> Failing
      <th id="r2"> Grade
      <td> Passing
    </tr>
  </tbody>
</table>

```

NOTE:

In the example above the [aria-describedby](#) attribute is used to explicitly associate the information with the table for assistive technology users.

Next to the table, in the same figure

EXAMPLE 451

```

<figure aria-labelledby="caption">
  <p>Characteristics are given in the second column, with the
  negative side in the left column and the positive side in the right
  column.</p>
  <table>
    <caption id="caption">Characteristics with positive and negative sides</caption>
    <thead>
      <tr>
        <th id="n"> Negative
        <th> Characteristic
        <th> Positive
      </tr>
    </thead>
    <tbody>
      <tr>
        <td headers="n r1"> Sad
        <th id="r1"> Mood
        <td> Happy
      </tr>
      <tr>
        <td headers="n r2"> Failing
        <th id="r2"> Grade
        <td> Passing
      </tr>
    </tbody>
  </table>
</figure>

```

NOTE:

The [<figure>](#) in this example has been labeled by the table [<caption>](#) using [aria-labelledby](#).

Authors may also use other techniques, or combinations of the above techniques, as appropriate.

NOTE:

Regardless of the method used to provide additional descriptive information for a `table`, if a `<table>` needs a caption, authors should use a `<caption>` element as it is the most robust method for providing an accessible caption for a `table`.

The best option, of course, rather than writing a description explaining the way the table is laid out, is to adjust the table such that no explanation is needed.

EXAMPLE 452

In the case of the table used in the examples above, a simple rearrangement of the table so that the headers are on the top and left sides removes the need for an explanation as well as removing the need for the use of `headers` attributes:

```
<table>
  <caption>Characteristics with positive and negative sides</caption>
  <thead>
    <tr>
      <th> Characteristic
      <th> Negative
      <th> Positive
    </tr>
  </thead>
  <tbody>
    <tr>
      <th> Mood
      <td> Sad
      <td> Happy
    </tr>
    <tr>
      <th> Grade
      <td> Failing
      <td> Passing
    </tr>
  </tbody>
</table>
```

4.9.1.2. Techniques for table design

Good table design is key to making tables more readable and usable.

In visual media, providing column and row borders and alternating row backgrounds can be very effective to make complicated tables more readable.

For tables with large volumes of numeric content, using monospaced fonts can help users see patterns, especially in situations where a user agent does not render the borders. (Unfortunately, for historical reasons, not rendering borders on tables is a common default.)

In speech media, table cells can be distinguished by reporting the corresponding headers before reading the cell's contents, and by allowing users to navigate the table in a grid fashion, rather than serializing the entire contents of the table in source order.

Authors are encouraged to use CSS to achieve these effects.

User agents are encouraged to render tables using these techniques whenever the page does not use CSS and the table is not classified as a layout table.

4.9.2. The caption element**Categories:**

None.

Contexts in which this element can be used:

As the first element child of a `<table>` element.

Content model:

Flow content, but with no descendant `<table>` elements.

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed [ARIA role attribute](#) values:

[Any role value.](#)

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles.](#)

DOM interface:

```
interface HTMLTableCaptionElement : HTMLElement {};
```

The `<caption>` element [represents](#) the title of the table that is its parent, if it has a parent and that is a `<table>` element.

The `<caption>` element takes part in the [table model](#).

When a `<table>` element is the only content in a `<figure>` element other than the `<figcaption>`, the `<caption>` element should be omitted in favor of the `<figcaption>`.

A caption can introduce context for a table, making it significantly easier to understand.

EXAMPLE 453

Consider, for instance, the following table:

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

In the abstract, this table is not clear. However, with a caption giving the table's number (for reference in the main prose) and explaining its use, it makes more sense:

```
<caption>
<p>Table 1.
<p>This table shows the total score obtained from rolling two
six-sided dice. The first row represents the value of the first die,
the first column the value of the second die. The total is given in
the cell that corresponds to the values of the two dice.
</caption>
```

This provides the user with more context:

Table 1. This table shows the total score obtained from rolling two six-sided dice. The first row represents the value of the first die, the first column the value of the second die. The total is given in the cell that corresponds to the values of the two dice.

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

4.9.3. The `colgroup` element

Categories:

None.

Contexts in which this element can be used:

As a child of a `<table>` element, after any `<caption>` elements and before any `<thead>`, `<tbody>`, `<tfoot>`, and `tr` elements.

Content model:

If the `span` attribute is present: [Nothing](#).

If the `span` attribute is absent: Zero or more `col` and `<template>` elements.

Tag omission in text/html:

A `<colgroup>` element's [end tag](#) may be omitted if the `<colgroup>` element is not immediately followed by a [space character](#) or a [comment](#).

Content attributes:

[Global attributes](#)

`span` - Number of columns spanned by the element

Allowed ARIA role attribute values:

None

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

DOM interface:

```
interface HTMLTableColElement : HTMLElement {
    attribute unsigned long span;
};
```

The `<colgroup>` element [represents](#) a [group](#) of one or more [columns](#) in the [table](#) that is its parent, if it has a parent and that is a `<table>` element.

If the `<colgroup>` element contains no `<col>` elements, then the element may have a `span` content attribute specified, whose value must be a [valid non-negative integer](#) greater than zero.

The `<colgroup>` element and its `span` attribute take part in the [table model](#).

The `span` IDL attribute must [reflect](#) the content attribute of the same name. The value must be [limited to only non-negative numbers greater than zero](#).

4.9.4. The `col` element

Categories:

None.

Contexts in which this element can be used:

As a child of a `<colgroup>` element that doesn't have a `span` attribute.

Content model:

[Nothing](#).

Tag omission in text/html:

No [end tag](#).

Content attributes:

[Global attributes](#)

`span`

Allowed ARIA role attribute values:

None

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

DOM interface:

[HTMLTableColElement](#), same as for [<colgroup>](#) elements. This interface defines one member, [](#).

If a [<col>](#) element has a parent and that is a [<colgroup>](#) element that itself has a parent that is a [<table>](#) element, then the [<col>](#) element [represents](#) one or more [columns](#) in the [column group](#) represented by that [<colgroup>](#).

The element may have a **span** content attribute specified, whose value must be a [valid non-negative integer](#) greater than zero.

The [<col>](#) element and its span attribute take part in the [table model](#).

The [span](#) IDL attribute must [reflect](#) the content attribute of the same name. The value must be [limited to only non-negative numbers greater than zero](#).

4.9.5. The `tbody` element

Categories:

None.

Contexts in which this element can be used:

As a child of a [<table>](#) element, after any [<caption>](#), [<colgroup>](#), and [<thead>](#) elements, but only if there are no [<tr>](#) elements that are children of the [<table>](#) element.

Content model:

Zero or more `tr` and [script-supporting elements](#).

Tag omission in text/html:

A [<tbody>](#) element's [start tag](#) may be omitted if the first thing inside the [<tbody>](#) element is a [<tr>](#) element, and if the element is not immediately preceded by a [<tbody>](#), [<thead>](#), or [<tfoot>](#) element whose [end tag](#) has been omitted. (It can't be omitted if the element is empty.). A [<tbody>](#) element's [end tag](#) may be omitted if the [<tbody>](#) element is immediately followed by a `tbody` or [<tfoot>](#) element, or if there is no more content in the parent element.

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[Any role value](#).

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLTableSectionElement : HTMLElement {
  [SameObject] readonly attribute HTMLCollection rows;
  HTMLElement insertRow(optional long index = -1);
  void deleteRow(long index);
};
```

The [HTMLTableSectionElement](#) interface is also used for [<thead>](#) and [<tfoot>](#) elements.

The [<tbody>](#) element [represents](#) a [block](#) of [rows](#) that consist of a body of data for the parent [<table>](#) element, if the [<tbody>](#) element has a parent and it is a [<table>](#).

The [<tbody>](#) element takes part in the [table model](#).

This definition is non-normative. Implementation requirements are given below this definition.

`tbody` . rows

Returns an HTMLCollection of the [<tr>](#) elements of the table section.

```
tr = tbody.insertRow( [ index ] )
```

Creates a [<tr>](#) element, inserts it into the table section at the position given by the argument, and returns the [<tr>](#).

The position is relative to the rows in the table section. The index -1, which is the default if the argument is omitted, is equivalent to inserting at the end of the table section.

If the given position is less than -1 or greater than the number of rows, throws an `IndexSizeError` exception.

```
tbody.deleteRow(index)
```

Removes the [<tr>](#) element with the given position in the table section.

The position is relative to the rows in the table section. The index -1 is equivalent to deleting the last row of the table section.

If the given position is less than -1 or greater than the index of the last row, or if there are no rows, throws an `IndexSizeError` exception.

The **rows** attribute must return an `HTMLCollection` rooted at the element, whose filter matches only `tr` elements that are children of the element.

The **insertRow**(*index*) method must, when invoked on an element *table section*, act as follows:

If *index* is less than -1 or greater than the number of elements in the **rows** collection, the method must throw an `IndexSizeError` exception.

If *index* is -1 or equal to the number of items in the **rows** collection, the method must create a [<tr>](#) element, append it to the element *table section*, and return the newly created [<tr>](#) element.

Otherwise, the method must create a [<tr>](#) element, insert it as a child of the *table section* element, immediately before the *index*th [<tr>](#) element in the **rows** collection, and finally must return the newly created [<tr>](#) element.

The **deleteRow**(*index*) method must, when invoked, act as follows:

If *index* is less than -1 or greater than the number of elements in the **rows** collection, the method must throw an `IndexSizeError` exception.

If *index* is -1, remove the last element in the **rows** collection from its parent.

Otherwise, remove the *index*th element in the **rows** collection from its parent.

§ 4.9.6. The **thead** element

Categories:

None.

Contexts in which this element can be used:

As a child of a [<table>](#) element, after any [<caption>](#), and `colgroup` elements and before any [<tbody>](#), [<tfoot>](#), and [<tr>](#) elements, but only if there are no other [<thead>](#) elements that are children of the [<table>](#) element.

Content model:

Zero or more `tr` and [script-supporting elements](#).

Tag omission in text/html:

A [<thead>](#) element's [end tag](#) may be omitted if the [<thead>](#) element is immediately followed by a `tbody` or [<tfoot>](#) element.

Content attributes:

[Global attributes](#)

Allowed [ARIA role attribute](#) values:

[Any role value](#).

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any `aria-*` attributes [applicable to the allowed roles](#).

DOM interface:

HTMLTableSectionElement, as defined for [<tbody>](#) elements.

The [<thead>](#) element [represents](#) the [block](#) of [rows](#) that consist of the column labels (headers) for the parent [<table>](#) element, if the [thead](#) element has a parent and it is a [<table>](#).

The [<thead>](#) element takes part in the [table model](#).

EXAMPLE 454

This example shows a [<thead>](#) element being used. Notice the use of both [th](#) and [<td>](#) elements in the [<thead>](#) element: the first row is the headers, and the second row is an explanation of how to fill in the table.

```
<table>
  <caption> School auction sign-up sheet </caption>
  <thead>
    <tr>
      <th><label for=e1>Name</label>
      <th><label for=e2>Product</label>
      <th><label for=e3>Picture</label>
      <th><label for=e4>Price</label>
    <tr>
      <td>Your name here
      <td>What are you selling?
      <td>Link to a picture
      <td>Your reserve price
  <tbody>
    <tr>
      <td>Ms Danus
      <td>Doughnuts
      <td>
      <td>$45
    <tr>
      <td><input id=e1 type=text name=who required form=f>
      <td><input id=e2 type=text name=what required form=f>
      <td><input id=e3 type=url name=pic form=f>
      <td><input id=e4 type=number step=0.01 min=0 value=0 required form=f>
  </tbody>
</table>
<form id=f action="/auction.cgi">
  <input type=button name=add value="Submit">
</form>
```

§ 4.9.7. The tfoot element**Categories:**

None.

Contexts in which this element can be used:

As a child of a [<table>](#) element, after any [<caption>](#), [<colgroup>](#), [<thead>](#), [<tbody>](#), and [<tr>](#) elements, but only if there are no other [<tfoot>](#) elements that are children of the [<table>](#) element.

Content model:

Zero or more [tr](#) and [script-supporting elements](#).

Tag omission in text/html:

A [<tfoot>](#) element's [end tag](#) may be omitted if the [<tfoot>](#) element is immediately followed by a [<tbody>](#) element, or if there is no more content in the parent element.

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[Any role value](#).

Allowed ARIA state and property attributes:[Global aria-* attributes](#)Any aria-* attributes [applicable to the allowed roles](#).**DOM interface:**HTMLTableSectionElement, as defined for [<tbody>](#) elements.

The [<tfoot>](#) element [represents](#) the [block](#) of [rows](#) that consist of the column summaries (footers) for the parent [<table>](#) element, if the [<tfoot>](#) element has a parent and it is a [<table>](#).

The [<tfoot>](#) element takes part in the [table model](#).

4.9.8. The tr element**Categories:**

None.

Contexts in which this element can be used:As a child of a [<thead>](#) element.As a child of a [<tbody>](#) element.As a child of a [<tfoot>](#) element.As a child of a [<table>](#) element, after any [<caption>](#), [<colgroup>](#), and [thead](#) elements, but only if there are no [<tbody>](#) elements that are children of the [<table>](#) element.**Content model:**Zero or more [<td>](#), [<th>](#), and [script-supporting elements](#).**Tag omission in text/html:**

A [<tr>](#) element's [end tag](#) may be omitted if the [<tr>](#) element is immediately followed by another [<tr>](#) element, or if there is no more content in the parent element.

Content attributes:[Global attributes](#)**Allowed ARIA role attribute values:**[Any role value](#).**Allowed ARIA state and property attributes:**[Global aria-* attributes](#)Any aria-* attributes [applicable to the allowed roles](#).**DOM interface:**

```
interface HTMLTableRowElement : HTMLElement {
  readonly attribute long rowIndex;
  readonly attribute long sectionRowIndex;
  [SameObject] readonly attribute HTMLCollection cells;
  HTMLElement insertCell(optional long index = -1);
  void deleteCell(long index);
};
```

The [<tr>](#) element [represents](#) a [row](#) of [cells](#) in a [table](#).

The [<tr>](#) element takes part in the [table model](#).

This definition is non-normative. Implementation requirements are given below this definition.

`tr` . `rowIndex`

Returns the position of the row in the table's rows list.

Returns -1 if the element isn't in a table.

`tr` . `sectionRowIndex`

Returns the position of the row in the table section's rows list.

Returns -1 if the element isn't in a table section.

`tr` . `cells`

Returns an `HTMLCollection` of the `td` and `<th>` elements of the row.

`cell` = `tr` . `insertCell`([`index`])

Creates a `<td>` element, inserts it into the table row at the position given by the argument, and returns the `<td>`.

The position is relative to the cells in the row. The index -1, which is the default if the argument is omitted, is equivalent to inserting at the end of the row.

If the given position is less than -1 or greater than the number of cells, throws an `IndexSizeError` exception.

`tr` . `deleteCell`(`index`)

Removes the `td` or `<th>` element with the given position in the row.

The position is relative to the cells in the row. The index -1 is equivalent to deleting the last cell of the row.

If the given position is less than -1 or greater than the index of the last cell, or if there are no cells, throws an `IndexSizeError` exception.

The **`rowIndex`** attribute must, if the element has a parent `<table>` element, or a parent `<tbody>`, `<thead>`, or `<tfoot>` element and a *grandparent* `<table>` element, return the index of the `<tr>` element in that `<table>` element's rows collection. If there is no such `<table>` element, then the attribute must return -1.

The **`sectionRowIndex`** attribute must, if the element has a parent `<table>`, `<tbody>`, `<thead>`, or `<tfoot>` element, return the index of the `<tr>` element in the parent element's rows collection (for tables, that's the `HTMLTableElement.rows` collection; for table sections, that's the `HTMLTableRowElement.rows` collection). If there is no such parent element, then the attribute must return -1.

The **`cells`** attribute must return an `HTMLCollection` rooted at the `<tr>` element, whose filter matches only `td` and `<th>` elements that are children of the `<tr>` element.

The **`insertCell( index )`** method must act as follows:

If `index` is less than -1 or greater than the number of elements in the `cells` collection, the method must throw an `IndexSizeError` exception.

If `index` is equal to -1 or equal to the number of items in `cells` collection, the method must create a `<td>` element, append it to the `<tr>` element, and return the newly created `td` element.

Otherwise, the method must create a `<td>` element, insert it as a child of the `<tr>` element, immediately before the `index`th `td` or `<th>` element in the `cells` collection, and finally must return the newly created `<td>` element.

The **`deleteCell( index )`** method must act as follows:

If `index` is less than -1 or greater than the number of elements in the `cells` collection, the method must throw an `IndexSizeError` exception.

If `index` is -1, remove the last element in the `cells` collection from its parent.

Otherwise, remove the `index`th element in the `cells` collection from its parent.

§ 4.9.9. The `td` element

Categories:

[Sectioning root.](#)

Contexts in which this element can be used:

As a child of a `<tr>` element.

Content model:

[Flow content.](#)

Tag omission in text/html:

A `<td>` element's [end tag](#) may be omitted if the `<td>` element is immediately followed by a `td` or `<th>` element, or if there is no more content in the parent element.

Content attributes:[Global attributes](#)

`colspan` - Number of columns that the cell is to span

`rowspan` - Number of rows that the cell is to span

`headers` - The header cells for this cell

Allowed [ARIA role attribute](#) values:

[Any role value.](#)

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles.](#)

DOM interface:

```
interface HTMLTableDataCellElement : HTMLTableCellElement {};
```

The `<td>` element [represents](#) a data [cell](#) in a table.

The `<td>` element and its `colspan`, `rowspan`, and `headers` attributes take part in the [table model](#).

User agents, especially in non-visual environments or where displaying the table as a 2D grid is impractical, may give the user context for the cell when rendering the contents of a cell; for instance, giving its position in the [table model](#), or listing the cell's header cells (as determined by the [algorithm for assigning header cells](#)). When a cell's header cells are being listed, user agents may use the value of `abbr` attributes on those header cells, if any, instead of the contents of the header cells themselves.

4.9.10. The `th` element**[Categories:](#)**

None.

[Contexts in which this element can be used:](#)

As a child of a `<tr>` element.

[Content model:](#)

[Flow content](#), but with no `<header>`, `<footer>`, [sectioning content](#), or [heading content](#) descendants

Tag omission in text/html:

A `<th>` element's [end tag](#) may be omitted if the `<th>` element is immediately followed by a `td` or `<th>` element, or if there is no more content in the parent element.

Content attributes:[Global attributes](#)

`colspan` - Number of columns that the cell is to span

`rowspan` - Number of rows that the cell is to span

`headers` - The headers for this cell

`scope` - Specifies which cells the header cell applies to

`abbr` - Alternative label to use for the header cell when referencing the cell in other contexts

Allowed [ARIA role attribute](#) values:

[Any role value.](#)

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles.](#)

DOM interface:

```
interface HTMLTableHeaderCellElement : HTMLTableCellElement {  
    attribute DOMString scope;  
    attribute DOMString abbr;  
};
```

The `<th>` element [represents](#) a header [cell](#) in a table.

The `<th>` element may have a **scope** content attribute specified. The scope attribute is an [enumerated attribute](#) with five states, four of which have explicit keywords:

The *row* keyword, which maps to the *row* state

The **row** state means the header cell applies to some of the subsequent cells in the same row(s).

The *col* keyword, which maps to the *column* state

The **column** state means the header cell applies to some of the subsequent cells in the same column(s).

The *rowgroup* keyword, which maps to the *row group* state

The **row group** state means the header cell applies to all the remaining cells in the row group. A `<th>` element's scope attribute must not be in the [row group](#) state if the element is not anchored in a [row group](#).

The *colgroup* keyword, which maps to the *column group* state

The **colgroup group** state means the header cell applies to all the remaining cells in the column group. A `<th>` element's scope attribute must not be in the [column group](#) state if the element is not anchored in a [column group](#).

The *auto* state

The **auto** state makes the header cell apply to a set of cells selected based on context.

The scope attribute's *missing value default* is the *auto* state.

The `<th>` element may have an **abbr** content attribute specified. Its value must be an alternative label for the header cell, to be used when referencing the cell in other contexts (e.g., when describing the header cells that apply to a data cell). It is typically an abbreviated form of the full header cell, but can also be an expansion, or merely a different phrasing.

The `<th>` element and its `colspan`, `rowspan`, `headers`, and `scope` attributes take part in the [table model](#).

The **scope** IDL attribute must [reflect](#) the content attribute of the same name, [limited to only known values](#).

The **abbr** IDL attribute must [reflect](#) the content attribute of the same name.

EXAMPLE 455

The following example shows how the `scope` attribute's `rowgroup` value affects which data cells a header cell applies to.

Here is a markup fragment showing a table:

NOTE:

The `<tbody>` elements in this example identify the range of the row groups.

```
<table>
  <caption>Measurement of legs and tails in Cats and English speakers</caption>
  <thead>
    <tr> <th> ID <th> Measurement <th> Average <th> Maximum
  </thead>
  <tbody>
    <tr> <td> <th scope=rowgroup> Cats <td> <td>
    <tr> <td> 93 <th scope=row> Legs <td> 3.5 <td> 4
    <tr> <td> 10 <th scope=row> Tails <td> 1 <td> 1
  </tbody>
  <tbody>
    <tr> <td> <th scope=rowgroup> English speakers <td> <td>
    <tr> <td> 32 <th scope=row> Legs <td> 2.67 <td> 4
    <tr> <td> 35 <th scope=row> Tails <td> 0.33 <td> 1
  </tbody>
</table>
```

This would result in the following table:

Measurement of legs and tails in Cats and English speakers

ID	Measurement	Average	Maximum
	Cats		
93	Legs	3.5	4
10	Tails	1	1
	English speakers		
32	Legs	2.67	4
35	Tails	0.33	1

The header cells in row 1 ("ID", "Measurement", "Average" and "Maximum") each apply only to the cells in their column.

The header cells with a `scope=rowgroup` ("Cats" and "English speakers") apply to all the cells in their row group other than the cells (to their left) in column 1:

The header "Cats" (row 2, column 2) applies to the headers "Legs" (row 3, column 2) and "Tails" (row 4, column 2) and to the data cells in rows 2, 3 and 4 of the "Average" and "Maximum" columns.

The header "English speakers" (row 5, column 2) applies to the headers "Legs" (row 6, column 2) and "Tails" (row 7, column 2) and to the data cells in rows 5, 6 and 7 of the "Average" and "Maximum" columns.

Each of the "Legs" and "Tails" header cells has a `scope=row` and therefore apply to the data cells (to the right) in their row, from the "Average" and "Maximum" columns.

ID	Measurement	Average	Maximum
	Cats		
93	Legs	3.5	4
10	Tails	1	1
	English speakers		
32	Legs	2.67	4
35	Tails	0.33	1

§ 4.9.11. Attributes common to `td` and `<th>` elements

The `<td>` and `<th>` elements may have a **colspan** content attribute specified, whose value must be a [valid non-negative integer](#) greater than zero.

The `<td>` and `<th>` elements may also have a **rowspan** content attribute specified, whose value must be a [valid non-negative integer](#). For this attribute, the value zero means that the cell is to span all the remaining rows in the row group.

These attributes give the number of columns and rows respectively that the cell is to span. These attributes must not be used to overlap cells, as described in the description of the [table model](#).



The `<td>` and `<th>` element may have a **headers** content attribute specified. The headers attribute, if specified, must contain a string consisting of an [unordered set of unique space-separated tokens](#) that are [case-sensitive](#), each of which must have the value of an [id](#) of a `<th>` element taking part in the same [table](#) as the `<td>` or `<th>` element (as defined by the [table model](#)).

A `<th>` element with [id](#) *id* is said to be *directly targeted* by all `td` and `<th>` elements in the same [table](#) that have headers attributes whose values include as one of their tokens the [ID](#) *id*. A `<th>` element *A* is said to be *targeted* by a `th` or `<td>` element *B* if either *A* is *directly targeted* by *B* or if there exists an element *C* that is itself *targeted* by the element *B* and *A* is *directly targeted* by *C*.

A `<th>` element must not be *targeted* by itself.

The colspan, rowspan, and headers attributes take part in the [table model](#).



The `td` and `<th>` elements implement interfaces that inherit from the `HTMLTableCellElement` interface:

```
interface HTMLTableCellElement : HTMLElement {
  attribute unsigned long colspan;
  attribute unsigned long rowspan;
  [PutForwards=value] readonly attribute DOMTokenList headers;
  readonly attribute long cellIndex;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

`cell` . `cellIndex`

Returns the position of the cell in the row's `cells` list. This does not necessarily correspond to the *x*-position of the cell in the table, since earlier cells might cover multiple rows or columns.

Returns -1 if the element isn't in a row.

The **colSpan** IDL attribute must [reflect](#) the `colspan` content attribute. Its default value is 1.

The **rowSpan** IDL attribute must [reflect](#) the `rowspan` content attribute. Its default value is 1.

The **headers** IDL attribute must [reflect](#) the content attribute of the same name.

The **cellIndex** IDL attribute must, if the element has a parent `<tr>` element, return the index of the cell's element in the parent element's `cells` collection. If there is no such parent element, then the attribute must return -1.

§ 4.9.12. Processing model

The various table elements and their content attributes together define the **table model**.

A **table** consists of cells aligned on a two-dimensional grid of **slots** with coordinates (x, y) . The grid is finite, and is either empty or has one or more slots. If the grid has one or more slots, then the x coordinates are always in the range $0 \leq x < x_{width}$, and the y coordinates are always in the range $0 \leq y < y_{height}$. If one or both of x_{width} and y_{height} are zero, then the table is empty (has no slots). Tables correspond to `<table>` elements.

A **cell** is a set of slots anchored at a slot $(cell_x, cell_y)$, and with a particular *width* and *height* such that the cell covers all the slots with coordinates (x, y) where $cell_x \leq x < cell_x + width$ and $cell_y \leq y < cell_y + height$. Cells can either be *data cells* or *header cells*. Data cells correspond to `<td>` elements, and header cells correspond to `<th>` elements. Cells of both types can have zero or more associated header cells.

It is possible, in certain error cases, for two cells to occupy the same slot.

A **row** is a complete set of slots from $x=0$ to $x=x_{width}-1$, for a particular value of y . Rows usually correspond to `<tr>` elements, though a [row group](#) can have some implied [rows](#) at the end in some cases involving [cells](#) spanning multiple rows.

A **column** is a complete set of slots from $y=0$ to $y=y_{height}-1$, for a particular value of x . Columns can correspond to `<col>` elements. In the absence of `<col>` elements, columns are implied.

A **row group** is a set of [rows](#) anchored at a slot $(0, group_y)$ with a particular *height* such that the row group covers all the slots with coordinates (x, y) where $0 \leq x < x_{width}$ and $group_y \leq y < group_y + height$. Row groups correspond to `<tbody>`, `<thead>`, and `<tfoot>` elements. Not every row is necessarily in a row group.

A **column group** is a set of [columns](#) anchored at a slot $(group_x, 0)$ with a particular *width* such that the column group covers all the slots with coordinates (x, y) where $group_x \leq x < group_x + width$ and $0 \leq y < y_{height}$. Column groups correspond to `<colgroup>` elements. Not every column is necessarily in a column group.

[Row groups](#) cannot overlap each other. Similarly, [column groups](#) cannot overlap each other.

A [cell](#) cannot cover slots that are from two or more [row groups](#). It is, however, possible for a cell to be in multiple [column groups](#). All the slots that form part of one cell are part of zero or one [row groups](#) and zero or more [column groups](#).

In addition to [cells](#), [columns](#), [rows](#), [row groups](#), and [column groups](#), [tables](#) can have a `<caption>` element associated with them. This gives the table a heading, or legend.

A **table model error** is an error with the data represented by table elements and their descendants. Documents must not have table model errors.

§ 4.9.12.1. Forming a table

User agents must use the following algorithm to determine

- which elements correspond to which slots in a [table](#) associated with a `<table>` element,
- the dimensions of the table (x_{width} and y_{height}), and
- if there are any [table model errors](#).

NOTE:

The algorithm selects the first `<caption>` encountered and assigns it as the caption for the table, and selects the first `<thead>` and processes it. Until there is a `<thead>`, `<tfoot>`, `<tbody>` or `<tr>` element, it processes any `<colgroup>` elements encountered, and any `<col>` children, to create column groups. Finally, from the first `<thead>`, `<tfoot>`, `<tbody>` or `<tr>` element encountered as a child of the `<table>` it processes those elements, moving the first `<tfoot>` encountered to the end of the table respectively.

1. Let x_{width} be zero.
2. Let y_{height} be zero.
3. Let *table footer* be null.
4. Let *table header* be null.
5. Let *the table* be the `<table>` element. The x_{width} and y_{height} variables give *the table*'s dimensions. *The table* is initially empty.
6. If the `<table>` element has no children elements, then return *the table* (which will be empty), and abort these steps.
7. Associate the first `<caption>` element child of the `<table>` element with *the table*. If there are no such children, then it has no associated `<caption>` element.
8. Let the *current element* be the first element child of the `<table>` element.

If a step in this algorithm ever requires the *current element* to be **advanced to the next child of the table** when there is no such next child, then the user agent must jump to the step labeled *end*, near the end of this algorithm.

9. While the *current element* is not one of the following elements, **advance** the *current element* to the next child of the table:
 - `<colgroup>`
 - `<thead>`
 - `<tbody>`
 - `<tfoot>`
 - `<tr>`
10. If the *current element* is a `<colgroup>`, follow these substeps:

1. *Column groups*: Process the *current element* according to the appropriate case below:

➡ **If the *current element* has any `<col>` element children**

Follow these steps:

1. Let x_{start} have the value of x_{width} .
2. Let the *current column* be the first `<col>` element child of the `<colgroup>` element.
3. *Columns*: If the *current column* `<col>` element has a span attribute, then parse its value using the [rules for parsing non-negative integers](#).

If the result of parsing the value is not an error or zero, then let *span* be that value.

Otherwise, if the `<col>` element has no span attribute, or if trying to parse the attribute's value resulted in an error or zero, then let *span* be 1.
4. Increase x_{width} by *span*.
5. Let the last *span* [columns](#) in *the table* correspond to the *current column* `<col>` element.
6. If *current column* is not the last `<col>` element child of the `<colgroup>` element, then let the *current column* be the next `<col>` element child of the `<colgroup>` element, and return to the step labeled *columns*.
7. Let all the last [columns](#) in *the table* from $x = x_{start}$ to $x = x_{width} - 1$ form a new [column group](#),

anchored at the slot $(x_{start}, 0)$, with width $x_{width} - x_{start}$, corresponding to the `<colgroup>` element.

↪ If the *current element* has no `<col>` element children

1. If the `<colgroup>` element has a `span` attribute, then parse its value using the [rules for parsing non-negative integers](#).

If the result of parsing the value is not an error or zero, then let *span* be that value.

Otherwise, if the `<colgroup>` element has no `span` attribute, or if trying to parse the attribute's value resulted in an error or zero, then let *span* be 1.

2. Increase x_{width} by *span*.

3. Let the last *span* [columns](#) in the *table* form a new [column group](#), anchored at the slot $(x_{width} - span, 0)$, with width *span*, corresponding to the `<colgroup>` element.

2. [Advance](#) the *current element* to the next child of the `<table>`.

3. While the *current element* is not one of the following elements, [advance](#) the *current element* to the next child of the *table*:

- `<colgroup>`
- `<thead>`
- `<tbody>`
- `<tfoot>`
- `<tr>`

4. If the *current element* is a `<colgroup>` element, jump to the step labeled *column groups* above.

11. Let $y_{current}$ be zero.

12. Let the *list of downward-growing cells* be an empty list.

13. *Rows*: While the *current element* is not one of the following elements, [advance](#) the *current element* to the next child of the *table*:

- `<thead>`
- `<tbody>`
- `<tfoot>`
- `<tr>`

Run the [algorithm for processing row groups](#) for the first `<thead>` child of the `<table>`.

14. If the *current element* is a `<tfoot>` and the value of *table footer* is null, then run the following substeps:

1. let *table footer* be the current element;
2. [advance](#) the *current element* to the next child of the `<table>`, and
3. return to the step labeled *rows*.

15. If the *current element* is a `<thead>` and the value of *table header* is null, then run the following substeps:

1. let *table header* be the current element;
2. [advance](#) the *current element* to the next child of the `<table>`, and
3. return to the step labeled *rows*.

16. If the *current element* is a `<tr>` then run the [algorithm for processing rows](#), [advance](#) the *current element* to the next child of the `<table>`, and return to the step labeled *rows*.

17. Run the [algorithm for ending a row group](#).

18. The *current element* is either a `<thead>`, `<tfoot>`, or a `<tbody>`.

Run the [algorithm for processing row groups](#).

19. [Advance](#) the *current element* to the next child of the [<table>](#).
20. Return to the step labeled *rows*.
21. *End*: run the [algorithm for processing row groups](#) to process *table footer*.
22. If there exists a [row](#) or [column](#) in *the table* containing only [slots](#) that do not have a [cell](#) anchored to them, then this is a [table model error](#).
23. Return *the table*.

The **algorithm for processing row groups**, which is invoked by the set of steps above for processing [<thead>](#), [<tbody>](#), and [<tfoot>](#) elements, is:

1. Let y_{start} have the value of y_{height} .
2. For each [<tr>](#) element that is a child of the element being processed, in tree order, run the [algorithm for processing rows](#).
3. If $y_{height} > y_{start}$, then let all the last [rows](#) in *the table* from $y = y_{start}$ to $y = y_{height} - 1$ form a new [row group](#), anchored at the slot with coordinate $(0, y_{start})$, with height $y_{height} - y_{start}$, corresponding to the element being processed.
4. Run the [algorithm for ending a row group](#).

The **algorithm for ending a row group**, which is invoked by the set of steps above when starting and ending a block of rows, is:

1. While $y_{current}$ is less than y_{height} , follow these steps:
 1. Run the [algorithm for growing downward-growing cells](#).
 2. Increase $y_{current}$ by 1.
2. Empty the *list of downward-growing cells*.

The **algorithm for processing rows**, which is invoked by the set of steps above for processing [<tr>](#) elements, is:

1. If y_{height} is equal to $y_{current}$, then increase y_{height} by
 1. ($y_{current}$ is never *greater* than y_{height} .)
2. Let $x_{current}$ be 0.
3. Run the [algorithm for growing downward-growing cells](#).
4. If the [<tr>](#) element being processed has no `td` or `th` element children, then increase $y_{current}$ by 1, abort this set of steps, and return to the algorithm above.
5. Let *current cell* be the first `td` or [<th>](#) element child in the [<tr>](#) element being processed.
6. *Cells*: While $x_{current}$ is less than x_{width} and the slot with coordinate $(x_{current}, y_{current})$ already has a cell assigned to it, increase $x_{current}$ by 1.
7. If $x_{current}$ is equal to x_{width} , increase x_{width} by 1. ($x_{current}$ is never *greater* than x_{width} .)
8. If the *current cell* has a `colspan` attribute, then [parse that attribute's value](#), and let *colspan* be the result.
If parsing that value failed, or returned zero, or if the attribute is absent, then let *colspan* be 1, instead.
9. If the *current cell* has a `rowspan` attribute, then [parse that attribute's value](#), and let *rowspan* be the result.
If parsing that value failed or if the attribute is absent, then let *rowspan* be 1, instead.
10. If *rowspan* is zero and the [<table>](#) element's [node document](#) is not set to [quirks mode](#), then let *cell grows downward* be true, and set *rowspan* to 1. Otherwise, let *cell grows downward* be false.

11. If $x_{width} < x_{current} + colspan$, then let x_{width} be $x_{current} + colspan$.
 12. If $y_{height} < y_{current} + rowspan$, then let y_{height} be $y_{current} + rowspan$.
 13. Let the slots with coordinates (x, y) such that $x_{current} \leq x < x_{current} + colspan$ and $y_{current} \leq y < y_{current} + rowspan$ be covered by a new [cell](#) c , anchored at $(x_{current}, y_{current})$, which has width $colspan$ and height $rowspan$, corresponding to the *current cell* element.
- If the *current cell* element is a [<th>](#) element, let this new cell c be a header cell; otherwise, let it be a data cell.
- To establish which header cells apply to the *current cell* element, use the [algorithm for assigning header cells](#) described in the next section.
- If any of the slots involved already had a [cell](#) covering them, then this is a [table model error](#). Those slots now have two cells overlapping.
14. If *cell grows downward* is true, then add the tuple $\{c, x_{current}, colspan\}$ to the *list of downward-growing cells*.
 15. Increase $x_{current}$ by $colspan$.
 16. If *current cell* is the last td or [<th>](#) element child in the [<tr>](#) element being processed, then increase $y_{current}$ by 1, abort this set of steps, and return to the algorithm above.
 17. Let *current cell* be the next td or [<th>](#) element child in the [<tr>](#) element being processed.
 18. Return to the step labeled *cells*.

When the algorithms above require the user agent to run the **algorithm for growing downward-growing cells**, the user agent must, for each $\{cell, cell_x, width\}$ tuple in the *list of downward-growing cells*, if any, extend the [cell](#) $cell$ so that it also covers the slots with coordinates $(x, y_{current})$, where $cell_x \leq x < cell_x + width$.

§ 4.9.12.2. Forming relationships between data cells and header cells

Each cell can be assigned zero or more header cells. The **algorithm for assigning header cells** to a cell *principal cell* is as follows.

1. Let *header list* be an empty list of cells.
2. Let $(principal_x, principal_y)$ be the coordinate of the slot to which the *principal cell* is anchored.
3. **↪ If the *principal cell* has a headers attribute specified**
 1. Take the value of the *principal cell*'s headers attribute and [split it on spaces](#), letting *id list* be the list of tokens obtained.
 2. For each token in the *id list*, if the first element in the [Document](#) with an [ID](#) equal to the token is a cell in the same [table](#), and that cell is not the *principal cell*, then add that cell to *header list*.
- ↪ If *principal cell* does not have a headers attribute specified**
 1. Let $principal_{width}$ be the width of the *principal cell*.
 2. Let $principal_{height}$ be the height of the *principal cell*.
 3. For each value of y from $principal_y$ to $principal_y + principal_{height} - 1$, run the [internal algorithm for scanning and assigning header cells](#), with the *principal cell*, the *header list*, the initial coordinate $(principal_x, y)$, and the increments $\Delta x = -1$ and $\Delta y = 0$.
 4. For each value of x from $principal_x$ to $principal_x + principal_{width} - 1$, run the [internal algorithm for scanning and assigning header cells](#), with the *principal cell*, the *header list*, the initial coordinate $(x, principal_y)$, and the increments $\Delta x = 0$ and $\Delta y = -1$.
 5. If the *principal cell* is anchored in a [row group](#), then add all header cells that are [row group headers](#) and are anchored in the same row group with an x -coordinate less than or equal to $principal_x + principal_{width} - 1$ and a y -coordinate less than or equal to $principal_y + principal_{height} - 1$ to

header list.

6. If the *principal cell* is anchored in a *column group*, then add all header cells that are *column group headers* and are anchored in the same column group with an *x*-coordinate less than or equal to $\text{principal}_x + \text{principal}_{width} - 1$ and a *y*-coordinate less than or equal to $\text{principal}_y + \text{principal}_{height} - 1$ to *header list*.

4. Remove all the *empty cells* from the *header list*.
5. Remove any duplicates from the *header list*.
6. Remove *principal cell* from the *header list* if it is there.
7. Assign the headers in the *header list* to the *principal cell*.

The **internal algorithm for scanning and assigning header cells**, given a *principal cell*, a *header list*, an initial co-ordinate (initial_x , initial_y), and Δx and Δy increments, is as follows:

1. Let *x* equal initial_x .
2. Let *y* equal initial_y .
3. Let *opaque headers* be an empty list of cells.
4. **↪ If *principal cell* is a header cell**
 Let *in header block* be true, and let *headers from current header block* be a list of cells containing just the *principal cell*.
- ↪ Otherwise**
 Let *in header block* be false and let *headers from current header block* be an empty list of cells.
5. *Loop*: Increment *x* by Δx ; increment *y* by Δy .

NOTE:

For each invocation of this algorithm, one of Δx and Δy will be -1, and the other will be 0.

6. If either *x* or *y* is less than 0, then abort this internal algorithm.
7. If there is no cell covering slot (*x*, *y*), or if there is more than one cell covering slot (*x*, *y*), return to the substep labeled *loop*.
8. Let *current cell* be the cell covering slot (*x*, *y*).
9. **↪ If *current cell* is a header cell**
 1. Set *in header block* to true.
 2. Add *current cell* to *headers from current header block*.
 3. Let *blocked* be false.
 4. **↪ If Δx is 0**
 If there are any cells in the *opaque headers* list anchored with the same *x*-coordinate as the *current cell*, and with the same width as *current cell*, then let *blocked* be true.

 If the *current cell* is not a *column header*, then let *blocked* be true.
 - ↪ If Δy is 0**
 If there are any cells in the *opaque headers* list anchored with the same *y*-coordinate as the *current cell*, and with the same height as *current cell*, then let *blocked* be true.

 If the *current cell* is not a *row header*, then let *blocked* be true.
 5. If *blocked* is false, then add the *current cell* to the *headers list*.
- ↪ If *current cell* is a data cell and *in header block* is true**

Set *in header block* to false. Add all the cells in *headers from current header block* to the *opaque headers* list, and empty the *headers from current header block* list.

10. Return to the step labeled *loop*.

A header cell anchored at the slot with coordinate (x, y) with width *width* and height *height* is said to be a **column header** if any of the following conditions are true:

- The cell's [scope](#) attribute is in the [column](#) state, or
- The cell's [scope](#) attribute is in the [auto](#) state, and there are no data cells in any of the cells covering slots with *y*-coordinates $y \dots y + \textit{height} - 1$.

A header cell anchored at the slot with coordinate (x, y) with width *width* and height *height* is said to be a **row header** if any of the following conditions are true:

- The cell's [scope](#) attribute is in the [row](#) state, or
- The cell's [scope](#) attribute is in the [auto](#) state, the cell is not a [column header](#), and there are no data cells in any of the cells covering slots with *x*-coordinates $x \dots x + \textit{width} - 1$.

A header cell is said to be a **column group header** if its [scope](#) attribute is in the [column group](#) state.

A header cell is said to be a **row group header** if its [scope](#) attribute is in the [row group](#) state.

A cell is said to be an **empty cell** if it contains no elements and its text content, if any, consists only of [White Space](#) characters.

§ 4.9.13. Examples

This section is non-normative.

The following shows how might one mark up the bottom part of table 45 of the *Smithsonian physical tables, Volume 71*:

```

<table>
  <caption>Specification values: <b>Steel</b>, <b>Castings</b>,
  Ann. A.S.T.M. A27-16, Class B;* P max. 0.06; S max. 0.05.</caption>
  <thead>
    <tr>
      <th rowspan=2>Grade.</th>
      <th rowspan=2>Yield Point.</th>
      <th colspan=2>Ultimate tensile strength</th>
      <th rowspan=2>Per cent elong. 50.8mm or 2 in.</th>
      <th rowspan=2>Per cent reduct. area.</th>
    </tr>
    <tr>
      <th>kg/mm<sup>2</sup></th>
      <th>lb/in<sup>2</sup></th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Hard</td>
      <td>0.45 ultimate</td>
      <td>56.2</td>
      <td>80,000</td>
      <td>15</td>
      <td>20</td>
    </tr>
    <tr>
      <td>Medium</td>
      <td>0.45 ultimate</td>
      <td>49.2</td>
      <td>70,000</td>
      <td>18</td>
      <td>25</td>
    </tr>
    <tr>
      <td>Soft</td>
      <td>0.45 ultimate</td>
      <td>42.2</td>
      <td>60,000</td>
      <td>22</td>
      <td>30</td>
    </tr>
  </tbody>
</table>

```

This table could look like this:

*Specification values: **Steel, Castings**, Ann. A.S.T.M. A27-16, Class B;* P max. 0.06; S max. 0.05.*

Grade.	Yield Point.	Ultimate tensile strength		Per cent elong. 50.8 mm or 2 in.	Per cent reduct. area.
		kg/mm ²	lb/in ²		
Hard	0.45 ultimate	56.2	80,000	15	20
Medium	0.45 ultimate	49.2	70,000	18	25
Soft	0.45 ultimate	42.2	60,000	22	30



The following shows how one might mark up the gross margin table on page 46 of Apple, Inc's 10-K filing for fiscal year 2008:

```

<table>
  <thead>
    <tr>
      <th>
        <th>2008
        <th>2007
        <th>2006
    <tbody>
      <tr>
        <th>Net sales
        <td>$ 32,479
        <td>$ 24,006
        <td>$ 19,315
      <tr>
        <th>Cost of sales
        <td> 21,334
        <td> 15,852
        <td> 13,717
      <tbody>
      <tr>
        <th>Gross margin
        <td>$ 11,145
        <td>$ 8,154
        <td>$ 5,598
      <tfoot>
      <tr>
        <th>Gross margin percentage
        <td>34.3%
        <td>34.0%
        <td>29.0%
    </table>

```

This table could look like this:

	2008	2007	2006
Net sales	\$ 32,479	\$ 24,006	\$ 19,315
Cost of sales	21,334	15,852	13,717
Gross margin	\$ 11,145	\$ 8,154	\$ 5,598
Gross margin percentage	34.3%	34.0%	29.0%



The following shows how one might mark up the operating expenses table from lower on the same page of that document:

```

<table>
  <colgroup> <col>
  <colgroup> <col> <col> <col>
  <thead>
    <tr> <th> <th>2008 <th>2007 <th>2006
  <tbody>
    <tr> <th scope=rowgroup> Research and development
      <td> $ 1,109 <td> $ 782 <td> $ 712
    <tr> <th scope=row> Percentage of net sales
      <td> 3.4% <td> 3.3% <td> 3.7%
  <tbody>
    <tr> <th scope=rowgroup> Selling, general, and administrative
      <td> $ 3,761 <td> $ 2,963 <td> $ 2,433
    <tr> <th scope=row> Percentage of net sales
      <td> 11.6% <td> 12.3% <td> 12.6%
</table>

```


This table could look like this:

	2008	2007	2006
Research and development	\$ 1,109	\$ 782	\$ 712
Percentage of net sales	3.4%	3.3%	3.7%
Selling, general, and administrative	\$ 3,761	\$ 2,963	\$ 2,433
Percentage of net sales	11.6%	12.3%	12.6%

§ 4.10. Forms

§ 4.10.1. Introduction

This section is non-normative.

A form is a component of a Web page that has form controls, such as text fields, buttons, checkboxes, range controls, or color pickers. A user can interact with such a form, providing data that can then be sent to the server for further processing (e.g., returning the results of a search or calculation). No client-side scripting is needed in many cases, though an API is available so that scripts can augment the user experience or use forms for purposes other than submitting data to a server.

Writing a form consists of several steps, which can be performed in any order: writing the user interface, implementing the server-side processing, and configuring the user interface to communicate with the server.

§ 4.10.1.1. Writing a form's user interface

This section is non-normative.

For the purposes of this brief introduction, we will create a pizza ordering form.

Any form starts with a `<form>` element, inside which are placed the controls. Most controls are represented by the `<input>` element, which by default provides a one-line text field. To label a control, the `<label>` element is used; the label text and the control itself go inside the `<label>` element. Each area within a form is typically represented using a `<div>` element. Putting this together, here is how one might ask for the customer's name:

```
<form>
  <div><label>Customer name: <input></label></div>
</form>
```

To let the user select the size of the pizza, we can use a set of radio buttons. Radio buttons also use the `<input>` element, this time with a `type` attribute with the value `radio`. To make the radio buttons work as a group, they are given a common name using the `name` attribute. To group a batch of controls together, such as, in this case, the radio buttons, one can use the `<fieldset>` element. The title of such a group of controls is given by the first element in the `<fieldset>`, which has to be a `<legend>` element.

```
<form>
  <div><label>Customer name: <input></label></div>
  <fieldset>
    <legend> Pizza Size </legend>
    <div><label> <input type=radio name=size> Small </label></div>
    <div><label> <input type=radio name=size> Medium </label></div>
    <div><label> <input type=radio name=size> Large </label></div>
  </fieldset>
</form>
```

NOTE:

Changes from the previous step are highlighted.

To pick toppings, we can use checkboxes. These use the `<input>` element with a `type` attribute with the value `checkbox`:

```

<form>
  <div><label>Customer name: <input></label></div>
  <fieldset>
    <legend> Pizza Size </legend>
    <div><label> <input type=radio name=size> Small </label></div>
    <div><label> <input type=radio name=size> Medium </label></div>
    <div><label> <input type=radio name=size> Large </label></div>
  </fieldset>
  <fieldset>
    <legend> Pizza Toppings </legend>
    <div><label> <input type=checkbox> Bacon </label></div>
    <div><label> <input type=checkbox> Extra Cheese </label></div>
    <div><label> <input type=checkbox> Onion </label></div>
    <div><label> <input type=checkbox> Mushroom </label></div>
  </fieldset>
</form>

```

The pizzeria for which this form is being written is always making mistakes, so it needs a way to contact the customer. For this purpose, we can use form controls specifically for telephone numbers ([<input>](#) elements with their [type](#) attribute set to `tel`) and e-mail addresses ([<input>](#) elements with their [type](#) attribute set to `email`):

```

<form>
  <div><label>Customer name: <input></label></div>
  <div><label>Telephone: <input type=tel></label></div>
  <div><label>E-mail address: <input type=email></label></div>
  <fieldset>
    <legend> Pizza Size </legend>
    <div><label> <input type=radio name=size> Small </label></div>
    <div><label> <input type=radio name=size> Medium </label></div>
    <div><label> <input type=radio name=size> Large </label></div>
  </fieldset>
  <fieldset>
    <legend> Pizza Toppings </legend>
    <div><label> <input type=checkbox> Bacon </label></div>
    <div><label> <input type=checkbox> Extra Cheese </label></div>
    <div><label> <input type=checkbox> Onion </label></div>
    <div><label> <input type=checkbox> Mushroom </label></div>
  </fieldset>
</form>

```

We can use an [<input>](#) element with its [type](#) attribute set to `time` to ask for a delivery time. Many of these form controls have attributes to control exactly what values can be specified; in this case, three attributes of particular interest are [min](#), [max](#), and [step](#). These set the minimum time, the maximum time, and the interval between allowed values (in seconds). This pizzeria only delivers between 11am and 9pm, and doesn't promise anything better than 15 minute increments, which we can mark up as follows:

```
<form>
  <div><label>Customer name: <input></label></div>
  <div><label>Telephone: <input type=tel></label></div>
  <div><label>E-mail address: <input type=email></label></div>
  <fieldset>
    <legend> Pizza Size </legend>
    <div><label> <input type=radio name=size> Small </label></div>
    <div><label> <input type=radio name=size> Medium </label></div>
    <div><label> <input type=radio name=size> Large </label></div>
  </fieldset>
  <fieldset>
    <legend> Pizza Toppings </legend>
    <div><label> <input type=checkbox> Bacon </label></div>
    <div><label> <input type=checkbox> Extra Cheese </label></div>
    <div><label> <input type=checkbox> Onion </label></div>
    <div><label> <input type=checkbox> Mushroom </label></div>
  </fieldset>
  <div><label>Preferred delivery time: <input type=time min="11:00" max="21:00" step="900">
</label></div>
</form>
```

The [<textarea>](#) element can be used to provide a free-form text field. In this instance, we are going to use it to provide a space for the customer to give delivery instructions:

```
<form>
  <div><label>Customer name: <input></label></div>
  <div><label>Telephone: <input type=tel></label></div>
  <div><label>E-mail address: <input type=email></label></div>
  <fieldset>
    <legend> Pizza Size </legend>
    <div><label> <input type=radio name=size> Small </label></div>
    <div><label> <input type=radio name=size> Medium </label></div>
    <div><label> <input type=radio name=size> Large </label></div>
  </fieldset>
  <fieldset>
    <legend> Pizza Toppings </legend>
    <div><label> <input type=checkbox> Bacon </label></div>
    <div><label> <input type=checkbox> Extra Cheese </label></div>
    <div><label> <input type=checkbox> Onion </label></div>
    <div><label> <input type=checkbox> Mushroom </label></div>
  </fieldset>
  <div><label>Preferred delivery time: <input type=time min="11:00" max="21:00" step="900">
</label></div>
  <div><label>Delivery instructions: <textarea></textarea></label></div>
</form>
```

Finally, to make the form submittable we use the [<button>](#) element:

```

<form>
  <div><label>Customer name: <input></label></div>
  <div><label>Telephone: <input type=tel></label></div>
  <div><label>E-mail address: <input type=email></label></div>
  <fieldset>
    <legend> Pizza Size </legend>
    <div><label> <input type=radio name=size> Small </label></div>
    <div><label> <input type=radio name=size> Medium </label></div>
    <div><label> <input type=radio name=size> Large </label></div>
  </fieldset>
  <fieldset>
    <legend> Pizza Toppings </legend>
    <div><label> <input type=checkbox> Bacon </label></div>
    <div><label> <input type=checkbox> Extra Cheese </label></div>
    <div><label> <input type=checkbox> Onion </label></div>
    <div><label> <input type=checkbox> Mushroom </label></div>
  </fieldset>
  <div><label>Preferred delivery time: <input type=time min="11:00" max="21:00" step="900">
</label></div>
  <div><label>Delivery instructions: <textarea></textarea></label></div>
  <div><button>Submit order</button></div>
</form>

```

§ 4.10.1.2. Implementing the server-side processing for a form

This section is non-normative.

The exact details for writing a server-side processor are out of scope for this specification. For the purposes of this introduction, we will assume that the script at `https://pizza.example.com/order.cgi` is configured to accept submissions using the `application/x-www-form-urlencoded` format, expecting the following parameters sent in an HTTP POST body:

custname

Customer's name

custtel

Customer's telephone number

custemail

Customer's e-mail address

size

The pizza size, either `small`, `medium`, or `large`

topping

A topping, specified once for each selected topping, with the allowed values being `bacon`, `cheese`, `onion`, and `mushroom`

delivery

The requested delivery time

comments

The delivery instructions

§ 4.10.1.3. Configuring a form to communicate with a server

This section is non-normative.

[Form submissions](#) are exposed to servers in a variety of ways, most commonly as HTTP GET or POST requests. To specify the exact method used, the `method` attribute is specified on the `<form>` element. This doesn't specify how the form data is encoded, though; to specify that, you use the `enctype` attribute. You also have to specify the [URL](#) of the service that will handle the submitted data, using the `action` attribute.

For each form control you want submitted, you then have to give a name that will be used to refer to the data in the submission. We already specified the name for the group of radio buttons; the same attribute ([name](#)) also specifies the submission name. Radio buttons can be distinguished from each other in the submission by giving them different values, using the `value` attribute.

Multiple controls can have the same name; for example, here we give all the checkboxes the same name, and the server distinguishes which checkbox was checked by seeing which values are submitted with that name — like the radio buttons, they are also given unique values with the `value` attribute.

Given the settings in the previous section, this all becomes:

```
<form method="post"
      enctype="application/x-www-form-urlencoded"
      action="https://pizza.example.com/order.cgi">
  <p><label>Customer name: <input name="custname"></label></p>
  <p><label>Telephone: <input type=tel name="custtel"></label></p>
  <p><label>E-mail address: <input type=email name="custemail"></label></p>
  <fieldset>
    <legend> Pizza Size </legend>
    <p><label> <input type=radio name=size value="small"> Small </label></p>
    <p><label> <input type=radio name=size value="medium"> Medium </label></p>
    <p><label> <input type=radio name=size value="large"> Large </label></p>
  </fieldset>
  <fieldset>
    <legend> Pizza Toppings </legend>
    <p><label> <input type=checkbox name="topping" value="bacon"> Bacon </label></p>
    <p><label> <input type=checkbox name="topping" value="cheese"> Extra Cheese </label></p>
    <p><label> <input type=checkbox name="topping" value="onion"> Onion </label></p>
    <p><label> <input type=checkbox name="topping" value="mushroom"> Mushroom </label></p>
  </fieldset>
  <p><label>Preferred delivery time: <input type=time min="11:00" max="21:00" step="900"
name="delivery"></label></p>
  <p><label>Delivery instructions: <textarea name="comments"></textarea></label></p>
  <p><button>Submit order</button></p>
</form>
```

NOTE:

There is no particular significance to the way some of the attributes have their values quoted and others don't. The HTML syntax allows a variety of equally valid ways to specify attributes, as discussed in [§8 The HTML syntax](#).

For example, if the customer entered "Denise Lawrence" as their name, "555-321-8642" as their telephone number, did not specify an e-mail address, asked for a medium-sized pizza, selected the Extra Cheese and Mushroom toppings, entered a delivery time of 7pm, and left the delivery instructions text field blank, the user agent would submit the following to the online Web service:

```
custname=Denise+Lawrence&custtel=555-321-8642&custemail=&size=medium&topping=cheese&
topping=mushroom&delivery=19%3A00&comments=
```

§ 4.10.1.4. Client-side form validation

This section is non-normative.

Forms can be annotated in such a way that the user agent will check the user's input before the form is submitted. The server still has to verify the input is valid (since hostile users can easily bypass the form validation), but it allows the user to avoid the wait incurred by having the server be the sole checker of the user's input.

The simplest annotation is the `required` attribute, which can be specified on `<input>` elements to indicate that the form is not to be submitted until a value is given. By adding this attribute to the customer name, pizza size, and delivery time fields, we allow the user agent to notify the user when the user submits the form without filling in those fields:

```

<form method="post"
      enctype="application/x-www-form-urlencoded"
      action="https://pizza.example.com/order.cgi">
  <p><label>Customer name: <input name="custname" required></label></p>
  <p><label>Telephone: <input type="tel" name="custtel"></label></p>
  <p><label>E-mail address: <input type="email" name="custemail"></label></p>
  <fieldset>
    <legend> Pizza Size </legend>
    <p><label> <input type="radio" name="size" required value="small"> Small </label></p>
    <p><label> <input type="radio" name="size" required value="medium"> Medium </label></p>
    <p><label> <input type="radio" name="size" required value="large"> Large </label></p>
  </fieldset>
  <fieldset>
    <legend> Pizza Toppings </legend>
    <p><label> <input type="checkbox" name="topping" value="bacon"> Bacon </label></p>
    <p><label> <input type="checkbox" name="topping" value="cheese"> Extra Cheese </label></p>
    <p><label> <input type="checkbox" name="topping" value="onion"> Onion </label></p>
    <p><label> <input type="checkbox" name="topping" value="mushroom"> Mushroom </label></p>
  </fieldset>
  <p><label>Preferred delivery time: <input type="time" min="11:00" max="21:00" step="900"
name="delivery" required></label></p>
  <p><label>Delivery instructions: <textarea name="comments"></textarea></label></p>
  <p><button>Submit order</button></p>
</form>

```

It is also possible to limit the length of the input, using the `maxlength` attribute. By adding this to the `<textarea>` element, we can limit users to 1000 characters, preventing them from writing huge essays to the busy delivery drivers instead of staying focused and to the point:

```

<form method="post"
      enctype="application/x-www-form-urlencoded"
      action="https://pizza.example.com/order.cgi">
  <p><label>Customer name: <input name="custname" required></label></p>
  <p><label>Telephone: <input type="tel" name="custtel"></label></p>
  <p><label>E-mail address: <input type="email" name="custemail"></label></p>
  <fieldset>
    <legend> Pizza Size </legend>
    <p><label> <input type="radio" name="size" required value="small"> Small </label></p>
    <p><label> <input type="radio" name="size" required value="medium"> Medium </label></p>
    <p><label> <input type="radio" name="size" required value="large"> Large </label></p>
  </fieldset>
  <fieldset>
    <legend> Pizza Toppings </legend>
    <p><label> <input type="checkbox" name="topping" value="bacon"> Bacon </label></p>
    <p><label> <input type="checkbox" name="topping" value="cheese"> Extra Cheese </label></p>
    <p><label> <input type="checkbox" name="topping" value="onion"> Onion </label></p>
    <p><label> <input type="checkbox" name="topping" value="mushroom"> Mushroom </label></p>
  </fieldset>
  <p><label>Preferred delivery time: <input type="time" min="11:00" max="21:00" step="900"
name="delivery" required></label></p>
  <p><label>Delivery instructions: <textarea name="comments" maxlength=1000></textarea>
</label></p>
  <p><button>Submit order</button></p>
</form>

```

NOTE:

When a form is submitted, invalid events are fired at each form control that is invalid, and then at the `<form>` element itself. This can be useful for displaying a summary of the problems with the form, since typically the browser itself will only report one problem at a time.

4.10.1.5. Enabling client-side automatic filling of form controls

This section is non-normative.

Some browsers attempt to aid the user by automatically filling form controls rather than having the user reenter their information each time. For example, a field asking for the user's telephone number can be automatically filled with the user's phone number.

To help the user agent with this, the [autocomplete](#) attribute can be used to describe the field's purpose. In the case of this form, we have three fields that can be usefully annotated in this way: the information about who the pizza is to be delivered to. Adding this information looks like this:

```
<form method="post"
  enctype="application/x-www-form-urlencoded"
  action="https://pizza.example.com/order.cgi">
  <p><label>Customer name: <input name="custname" required autocomplete="shipping name">
</label></p>
  <p><label>Telephone: <input type="tel" name="custtel" autocomplete="shipping tel"></label></p>
  <p><label>E-mail address: <input type="email" name="custemail" autocomplete="shipping email">
</label></p>
  <fieldset>
    <legend> Pizza Size </legend>
    <p><label> <input type="radio" name="size" required value="small"> Small </label></p>
    <p><label> <input type="radio" name="size" required value="medium"> Medium </label></p>
    <p><label> <input type="radio" name="size" required value="large"> Large </label></p>
  </fieldset>
  <fieldset>
    <legend> Pizza Toppings </legend>
    <p><label> <input type="checkbox" name="topping" value="bacon"> Bacon </label></p>
    <p><label> <input type="checkbox" name="topping" value="cheese"> Extra Cheese </label></p>
    <p><label> <input type="checkbox" name="topping" value="onion"> Onion </label></p>
    <p><label> <input type="checkbox" name="topping" value="mushroom"> Mushroom </label></p>
  </fieldset>
  <p><label>Preferred delivery time: <input type="time" min="11:00" max="21:00" step="900"
name="delivery" required></label></p>
  <p><label>Delivery instructions: <textarea name="comments" maxlength=1000></textarea>
</label></p>
  <p><button>Submit order</button></p>
</form>
```

4.10.1.6. Improving the user experience on mobile devices

This section is non-normative.

Some devices, in particular those with on-screen keyboards and those in locales with languages with many characters (e.g., Japanese), can provide the user with multiple input modalities. For example, when typing in a credit card number the user may wish to only see keys for digits 0-9, while when typing in their name they may wish to see a form field that by default capitalizes each word.

Using the `inputmode` attribute we can select appropriate input modalities:

```

<form method="post"
      enctype="application/x-www-form-urlencoded"
      action="https://pizza.example.com/order.cgi">
  <p><label>Customer name: <input name="custname" required autocomplete="shipping name"
inputmode="latin-name"></label></p>
  <p><label>Telephone: <input type=tel name="custtel" autocomplete="shipping tel"></label></p>
  <p><label>E-mail address: <input type=email name="custemail" autocomplete="shipping email">
</label></p>
  <fieldset>
    <legend> Pizza Size </legend>
    <p><label> <input type=radio name=size required value="small"> Small </label></p>
    <p><label> <input type=radio name=size required value="medium"> Medium </label></p>
    <p><label> <input type=radio name=size required value="large"> Large </label></p>
  </fieldset>
  <fieldset>
    <legend> Pizza Toppings </legend>
    <p><label> <input type=checkbox name="topping" value="bacon"> Bacon </label></p>
    <p><label> <input type=checkbox name="topping" value="cheese"> Extra Cheese </label></p>
    <p><label> <input type=checkbox name="topping" value="onion"> Onion </label></p>
    <p><label> <input type=checkbox name="topping" value="mushroom"> Mushroom </label></p>
  </fieldset>
  <p><label>Preferred delivery time: <input type=time min="11:00" max="21:00" step="900"
name="delivery" required></label></p>
  <p><label>Delivery instructions: <textarea name="comments" maxlength=1000 inputmode="latin-
prose"></textarea></label></p>
  <p><button>Submit order</button></p>
</form>

```

§ 4.10.1.7. The difference between the field type, the autofill field name, and the input modality

This section is non-normative.

The `type`, `autocomplete`, and `inputmode` attributes can seem confusingly similar. For instance, in all three cases, the string "email" is a valid value. This section attempts to illustrate the difference between the three attributes and provides advice suggesting how to use them.

The `type` attribute on `<input>` elements decides what kind of control the user agent will use to expose the field. Choosing between different values of this attribute is the same choice as choosing whether to use an `<input>` element, a `<textarea>` element, a `<select>` element, a `<keygen>` element, etc.

The `autocomplete` attribute, in contrast, describes what the value that the user will enter actually represents. Choosing between different values of this attribute is the same choice as choosing what the label for the element will be.

First, consider telephone numbers. If a page is asking for a telephone number from the user, the right form control to use is `<input type=tel>`. However, which `autocomplete` value to use depends on which phone number the page is asking for, whether they expect a telephone number in the international format or just the local format, and so forth.

For example, a page that forms part of a checkout process on an e-commerce site for a customer buying a gift to be shipped to a friend might need both the buyer's telephone number (in case of payment issues) and the friend's telephone number (in case of delivery issues). If the site expects international phone numbers (with the country code prefix), this could thus look like this:

```

<p><label>Your phone number: <input type=tel name=custtel autocomplete="billing tel"></label>
<p><label>Recipient's phone number: <input type=tel name=shiptel autocomplete="shipping
tel"></label>
<p>Please enter complete phone numbers including the country code prefix, as in "+1 555 123
4567".

```

But if the site only supports British customers and recipients, it might instead look like this (notice the use of `tel-national` rather than `tel`):


```

<p><label>Your phone number: <input type=tel name=cussttel autocomplete="billing tel-national">
</label>
<p><label>Recipient's phone number: <input type=tel name=shiptel autocomplete="shipping tel-
national"></label>
<p>Please enter complete UK phone numbers, as in "(01632) 960 123".

```

Now, consider a person's preferred languages. The right `autocomplete` value is `language`. However, there could be a number of different form controls used for the purpose: a free text field (`<input type=text>`), a drop-down list (`<select>`), radio buttons (`<input type=radio>`), etc. It only depends on what kind of interface is desired.

The `inputmode` decides what kind of input modality (e.g., keyboard) to use, when the control is a free-form text field.

Consider names. If a page just wants one name from the user, then the relevant control is `<input type=text>`. If the page is asking for the user's full name, then the relevant `autocomplete` value is `name`. But if the user is Japanese, and the page is asking for the user's Japanese name and the user's romanized name, then it would be helpful to the user if the first field defaulted to a Japanese input modality, while the second defaulted to a Latin input modality (ideally with automatic capitalization of each word). This is where the `inputmode` attribute can help:

```

<p><label>Japanese name: <input name="j" type="text" autocomplete="section-jp name"
inputmode="kana"></label>
<label>Romanized name: <input name="e" type="text" autocomplete="section-en name"
inputmode="latin-name"></label>

```

In this example, the `"section-"` keywords in the `autocomplete` attributes' values tell the user agent that the two fields expect *different* names. Without them, the user agent could automatically fill the second field with the value given in the first field when the user gave a value to the first field.

NOTE:

The `"-jp"` and `"-en"` parts of the keywords are opaque to the user agent; the user agent cannot guess, from those, that the two names are expected to be in Japanese and English respectively.

4.10.1.8. Date, time, and number formats

This section is non-normative.

In this pizza delivery example, the times are specified in the format `"HH:MM"`: two digits for the hour, in 24-hour format, and two digits for the time. (Seconds could also be specified, though they are not necessary in this example.)

In some locales, however, times are often expressed differently when presented to users. For example, in the United States, it is still common to use the 12-hour clock with an am/pm indicator, as in `"2pm"`. In France, it is common to use the 24-hour clock, and separate the hours from the minutes using an `"h"` character, as in `"14h00"`.

Similar issues exist with dates, with the added complication that even the order of the components is not always consistent — for example, in Cyprus the first of February 2003 would typically be written `"1/2/03"`, while that same date in Japan would typically be written as `"2003年02月01日"`.

The same applies to the way numbers are written in different places. For example, in some locales, such as US English, `"1,234"` usually means "one thousand two hundred and thirty-four", while `"1.234"` means "one and two hundred and thirty-four thousandths", or "one point two three four", while in many other locales the meanings are exactly reversed.

The format used "on the wire", i.e., in HTML markup and as the values for [form submissions](#), is intended to be computer-readable and consistent irrespective of the user's locale, to allow scripts in pages and on servers to process times, dates, and numbers in a consistent manner with minimal work.

In HTML markup and form submission dates and times are always written in a locale-neutral format derived from the ISO-8601 standard. For example, dates are generally in a format such as `2016-06-19`.

Likewise, in markup and [form submission](#) numbers are always written without grouping separators, and a FULL STOP character `"."` `@@` as a decimal separator.

The time, date, or number may be translated to the user's preferred presentation (based on expressed preferences or on the locale of the page itself), before being displayed to the user. Similarly, user agents may allow a user to input a time, date, or number using their preferred format, then converts it back to the wire format before putting it in the DOM or submitting it.

This allows scripts in pages and on servers to process times, dates, and numbers in a consistent manner without needing to support dozens of different formats, while still supporting the users' needs.

NOTE:

See also the [implementation notes](#) regarding localization of form controls.

§ 4.10.2. Categories

Mostly for historical reasons, elements in this section fall into several overlapping (but subtly different) categories in addition to the usual ones like [flow content](#), [phrasing content](#), and [interactive content](#).

A number of the elements are **form-associated elements**, which means they can have a [form owner](#).

⇒ [<button>](#), [<fieldset>](#), [<input>](#), [<keygen>](#), [<label>](#), [<object>](#), [<output>](#), [<select>](#), [<textarea>](#), [](#)

The [form-associated elements](#) fall into several subcategories:

Listed elements

Denotes elements that are listed in the `form.elements` and `fieldset.elements` APIs.

⇒ [<button>](#), [<fieldset>](#), [<input>](#), [<keygen>](#), [<object>](#), [<output>](#), [<select>](#), [<textarea>](#)

Submittable elements

Denotes elements that can be used for [constructing the form data set](#) when a `<form>` element is [submitted](#).

⇒ [<button>](#), [<input>](#), [<keygen>](#), [<object>](#), [<select>](#), [<textarea>](#)

Some [submittable elements](#) can be, depending on their attributes, **buttons**. The prose below defines when an element is a button. Some buttons are specifically **submit buttons**.

Resettable elements

Denotes elements that can be affected when a `<form>` element is [reset](#).

⇒ [<input>](#), [<keygen>](#), [<output>](#), [<select>](#), [<textarea>](#)

Reassociateable elements

Denotes elements that have a `form` content attribute, and a matching `form` IDL attribute, that allow authors to specify an explicit [form owner](#).

⇒ [<button>](#), [<fieldset>](#), [<input>](#), [<keygen>](#), [<object>](#), [<output>](#), [<select>](#), [<textarea>](#)

Some elements, not all of them [form-associated](#), are categorized as **labelable elements**. These are elements that can be associated with a `<label>` element.

⇒ [<button>](#), [<input>](#) (if the `type` attribute is *not* in the [hidden](#) state), [<keygen>](#), [<meter>](#), [<output>](#), [<progress>](#), [<select>](#), [<textarea>](#)

The following table is non-normative and summarizes the above categories of form elements:

	form-associated	listed	submittable	resettable	reassociateable	labelable
	can have a form owner	listed in the form.elements and fieldset.elements APIs	can be used for constructing the form data set when a form element is submitted	can be affected when a form element is reset	have a <code>form</code> attribute (allows authors to specify an explicit form owner)	can be associated with a <code><label></code> element
<input>	yes	yes	yes	yes	yes	yes (except "hidden")

	form-associated	listed	submittable	resettable	reassociateable	labelable
<a href="#<button>"><code><button></code>	yes	yes	yes	no	yes	yes
<a href="#<select>"><code><select></code>	yes	yes	yes	yes	yes	yes
<a href="#<textarea>"><code><textarea></code>	yes	yes	yes	yes	yes	yes
<a href="#<fieldset>"><code><fieldset></code>	yes	yes	no	no	yes	no
<a href="#<output>"><code><output></code>	yes	yes	no	yes	yes	yes
<a href="#<object>"><code><object></code>	yes	yes	yes	no	yes	no
<a href="#<meter>"><code><meter></code>	no	no	no	no	no	yes
<a href="#<progress>"><code><progress></code>	no	no	no	no	no	yes
<a href="#<label>"><code><label></code>	yes	no	no	no	no	no
<code></code>	yes	no	no	no	no	no

§ 4.10.3. The `form` element

Categories:

[Flow content.](#)

[Palpable content.](#)

Contexts in which this element can be used:

Where [flow content](#) is expected.

Content model:

[Flow content](#), but with no `<form>` element descendants.

Tag omission in text/html:

Neither tag is omissible.

Content attributes:

[Global attributes](#)

[accept-charset](#) - Character encodings to use for [§4.10.22 Form submission](#)

[action](#) - URL to use for [§4.10.22 Form submission](#)

[autocomplete](#) - Default setting for autofill feature for controls in the form

[enctype](#) - Form data set encoding type to use for [§4.10.22 Form submission](#)

[method](#) - HTTP method to use for [§4.10.22 Form submission](#)

[name](#) - Name of form to use in the [document.forms](#) API

[novalidate](#) - Bypass form control validation for [§4.10.22 Form submission](#)

[target](#) - [browsing context](#) for [§4.10.22 Form submission](#)

Allowed ARIA role attribute values:

[Any role value.](#)

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any `aria-*` attributes [applicable to the allowed roles](#).

DOM interface:

```
[OverrideBuiltins]
interface HTMLFormElement : HTMLElement {
    attribute DOMString acceptCharset;
    attribute DOMString action;
    attribute DOMString autocomplete;
    attribute DOMString enctype;
    attribute DOMString encoding;
    attribute DOMString method;
    attribute DOMString name;
    attribute boolean noValidate;
    attribute DOMString target;

    [SameObject] readonly attribute HTMLFormControlsCollection elements;
    readonly attribute unsigned long length;
    getter Element (unsigned long index);
    getter (RadioNodeList or Element) (DOMString name);

    void submit();
    void reset();
    boolean checkValidity();
    boolean reportValidity();
};
```

The `<form>` element [represents](#) a collection of [form-associated elements](#), some of which can represent editable values that can be submitted to a server for processing.

The **accept-charset** content attribute gives the character encodings that are to be used for the submission. If specified, the value must be an [ordered set of unique space-separated tokens](#) that are [ASCII case-insensitive](#), and each token must be an [ASCII case-insensitive](#) match for one of the [labels](#) of an [ASCII-compatible encoding](#). [\[ENCODING\]](#)

The **name** content attribute represents the `<form>`'s name within the [forms](#) collection. The value must not be the empty string, and the value must be unique amongst the `<form>` elements in the [forms](#) collection that it is in, if any.

The **autocomplete** content attribute is an [enumerated attribute](#). The attribute has two states. The **on** keyword maps to the **on** state, and the **off** keyword maps to the **off** state. The attribute may also be omitted. The *missing value default* is the **on** state. The **off** state indicates that by default, form controls in the form will have their [autofill field name](#) set to "off"; the **on** state indicates that by default, form controls in the form will have their [autofill field name](#) set to "on".

The **action**, **enctype**, **method**, **enctype**, **novalidate**, and **target** attributes are [attributes for form submission](#).

This definition is non-normative. Implementation requirements are given below this definition.

`form . elements`

Returns an [HTMLFormControlsCollection](#) of the form controls in the form (excluding image buttons for historical reasons).

`form . length`

Returns the number of form controls in the form (excluding image buttons for historical reasons).

`form [ index ]`

Returns the *index*th element in the form (excluding image buttons for historical reasons).

`form [ name ]`

Returns the form control (or, if there are several, a [RadioNodeList](#) of the form controls) in the form with the given **ID** or **name** (excluding image buttons for historical reasons); or, if there are none, returns the `<img>` element with the given ID.

Once an element has been referenced using a particular name, that name will continue being available as a way to reference that element in this method, even if the element's actual **ID** or **name** changes, for as long as the element remains in the [Document](#).

If there are multiple matching items, then a [RadioNodeList](#) object containing all those elements is returned.

`form . submit()`

Submits the form.

`form . reset()`

Resets the form.

`form . checkValidity()`

Returns true if the form’s controls are all valid; otherwise, returns false.

`form . reportValidity()`

Returns true if the form’s controls are all valid; otherwise, returns false and informs the user.

The **autocomplete** IDL attribute must [reflect](#) the content attribute of the same name, [limited to only known values](#).

The **name** IDL attribute must [reflect](#) the content attribute of the same name.

The **acceptCharset** IDL attribute must [reflect](#) the [accept-charset](#) content attribute.



The **elements** IDL attribute must return an [HTMLFormControlsCollection](#) rooted at the `<form>` element’s [home subtree’s root element](#), whose filter matches [listed elements](#) whose [form owner](#) is the `<form>` element, with the exception of `<input>` elements whose [type](#) attribute is in the [image button](#) state, which must, for historical reasons, be excluded from this particular collection.

The **length** IDL attribute must return the number of nodes [represented](#) by the **elements** collection.

The [supported property indices](#) at any instant are the indices supported by the object returned by the **elements** attribute at that instant.

When a `<form>` element is **indexed for indexed property retrieval**, the user agent must return the value returned by the **item** method on the **elements** collection, when invoked with the given index as its argument.



Each `<form>` element has a mapping of names to elements called the **past names map**. It is used to persist names of controls even when they change names.

The [supported property names](#) consist of the names obtained from the following algorithm, in the order obtained from this algorithm:

- Let *sourced names* be an initially empty ordered list of tuples consisting of a string, an element, a source, where the source is either *id*, *name*, or *past*, and, if the source is *past*, an age.
- For each [listed element candidate](#) whose [form owner](#) is the `<form>` element, with the exception of any `<input>` elements whose [type](#) attribute is in the [image button](#) state, run these substeps:
 - If *candidate* has an [id](#) attribute, add an entry to *sourced names* with that [id](#) attribute’s value as the string, *candidate* as the element, and *id* as the source.
 - If *candidate* has a [name](#) attribute, add an entry to *sourced names* with that [name](#) attribute’s value as the string, *candidate* as the element, and *name* as the source.
- For each `<img>` element *candidate* whose [form owner](#) is the `<form>` element, run these substeps:
 - If *candidate* has an [id](#) attribute, add an entry to *sourced names* with that [id](#) attribute’s value as the string, *candidate* as the element, and *id* as the source.
 - If *candidate* has a [name](#) attribute, add an entry to *sourced names* with that [name](#) attribute’s value as the string, *candidate* as the element, and *name* as the source.
- For each entry *past entry* in the [past names map](#) add an entry to *sourced names* with the *past entry*’s name as the

string, *past entry*'s element as the element, *past* as the source, and the length of time *past entry* has been in the [past names map](#) as the age.

- Sort *sourced names* by [tree order](#) of the element entry of each tuple, sorting entries with the same element by putting entries whose source is *id* first, then entries whose source is *name*, and finally entries whose source is *past*, and sorting entries with the same element and source by their age, oldest first.
- Remove any entries in *sourced names* that have the empty string as their name.
- Remove any entries in *sourced names* that have the same name as an earlier entry in the map.
- Return the list of names from *sourced names*, maintaining their relative order.

The properties exposed in this way must be [unenumerable](#).

When a [<form>](#) element is **indexed for named property retrieval**, the user agent must run the following steps:

- Let *candidates* be a [live](#) `RadioNodeList` object containing all the [listed elements](#) whose [form owner](#) is the [<form>](#) element that have either an `id` attribute or a [name](#) attribute equal to *name*, with the exception of [<input>](#) elements whose [type](#) attribute is in the [Image Button](#) state, in [tree order](#).
- If *candidates* is empty, let *candidates* be a [live](#) `RadioNodeList` object containing all the [](#) elements that are descendants of the [<form>](#) element and that have either an `id` attribute or a [name](#) attribute equal to *name*, in [tree order](#).
- If *candidates* is empty, *name* is the name of one of the entries in the [<form>](#) element's [past names map](#): return the object associated with *name* in that map.
- If *candidates* contains more than one node, return *candidates* and abort these steps.
- Otherwise, *candidates* contains exactly one node. Add a mapping from *name* to the node in *candidates* in the [<form>](#) element's [past names map](#), replacing the previous entry with the same name, if any.
- Return the node in *candidates*.

If an element listed in a [<form>](#) element's [past names map](#) changes [form owner](#), then its entries must be removed from that map.



The **submit()** method, when invoked, must [submit](#) the [<form>](#) element from the [<form>](#) element itself, with the *submitted from submit() method* flag set.

The **reset()** method, when invoked, must run the following steps:

- If the [<form>](#) element is marked as *locked for reset*, then abort these steps.
- Mark the [<form>](#) element as **locked for reset**.
- [Reset](#) the [<form>](#) element.
- Unmark the [<form>](#) element as *locked for reset*.

If the **checkValidity()** method is invoked, the user agent must [statically validate the constraints](#) of the [<form>](#) element, and return true if the constraint validation return a *positive* result, and false if it returned a *negative* result.

If the **reportValidity()** method is invoked, the user agent must [interactively validate the constraints](#) of the [<form>](#) element, and return true if the constraint validation return a *positive* result, and false if it returned a *negative* result.

EXAMPLE 456

This example shows two search forms:

```
<form action="https://www.google.com/search" method="get">
  <label>Google: <input type="search" name="q"></label> <input type="submit"
value="Search...">
</form>
<form action="https://www.bing.com/search" method="get">
  <label>Bing: <input type="search" name="q"></label> <input type="submit" value="Search...">
</form>
```

§ 4.10.4. The `label` element

Categories:

[Flow content](#),
[Phrasing content](#),
[Interactive content](#),
[Form-associated element](#),
[Palpable content](#).

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content](#), but with no descendant [labelable elements](#) unless it is the element's [labeled control](#), and no descendant `<label>` elements.

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)
[for](#) - Associate the label with form control

Allowed ARIA role attribute values:

None

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

DOM interface:

```
interface HTMLLabelElement : HTMLElement {
  readonly attribute HTMLFormElement? form;
  attribute DOMString htmlFor;
  readonly attribute HTMLElement? control;
};
```

The `<label>` element [represents](#) a caption in a user interface. The caption can be associated with a specific form control, known as the `<label>` element's **labeled control**, either using the [for](#) attribute, or by putting the form control inside the `<label>` element itself.

Except where otherwise specified by the following rules, a `<label>` element has no [labeled control](#).

The **for** attribute may be specified to indicate a form control with which the caption is to be associated. If the attribute is specified, the attribute's value must be the [ID](#) of a [labelable element](#) in the same [Document](#) as the `<label>` element. If the attribute is specified and there is an element in the [Document](#) whose [ID](#) is equal to the value of the [for](#) attribute, and the first such element is a [labelable element](#), then that element is the `<label>` element's [labeled control](#).

EXAMPLE 457

The following example shows the use of a [for](#) attribute, to associate [<label>](#)s which do not contain the element they label.

```
<form>
  <table>
    <caption>Example, <label>'s for attribute</caption>
    <tr>
      <th><label for="name">Customer name: </label></th>
      <td><input name="name" id="name"></td>
    </tr>
  </table>
</form>
```

Note that the `id` attribute is required to associate the [for](#) attribute, while the [name](#) attribute is required so the value of the input will be submitted as part of the form.

If the [for](#) attribute is not specified, but the [<label>](#) element has a [labelable element](#) descendant, then the first such descendant in [tree order](#) is the [<label>](#) element's [labeled control](#).

The [<label>](#) element's [activation behavior](#) should match the platform's label behavior. Similarly, any additional presentation hints should match the platform's label presentation.

EXAMPLE 458

On many platforms activating a checkbox label checks the checkbox, while activating a text input's label focuses the input. Clicking the [<label>](#) "Lost" in the following snippet could trigger the user agent to [run synthetic click activation steps](#) on the checkbox, as if the element itself had been triggered by the user, while clicking the [<label>](#) "Where?" would [queue a task](#) that runs the [focusing steps](#) for the element to the text input:

```
<label><input type="checkbox" name="lost"> Lost</label><br> <label>Where? <input type="text"
name="where"></label>
```

If a [<label>](#) element has [interactive content](#) other than its [labeled control](#), the [activation behavior](#) of the [<label>](#) element for events targeted at those [interactive content](#) descendants and any descendants of those must be to do nothing.

EXAMPLE 459

In the following example, clicking on the link does not toggle the checkbox, even if the platform normally toggles a checkbox when clicking on a label. Instead, clicking the link triggers the normal [activation behavior](#) of following the link.

```
<!-- bad example - link inside label reduces checkbox activation area -->
<label><input type=checkbox name=tac>I agree to <a href="tandc.html">the terms and
conditions</a></label>
```

NOTE:

The ability to click or press a [<label>](#) to trigger an event on a control provides usability and accessibility benefits by increasing the hit area of a control, making it easier for a user to operate. These benefits may be lost or reduced, if the [<label>](#) element contains an element with its own activation behavior, such as a link:

EXAMPLE 460

```
<!-- bad example - all label text inside the link reduces activation area to checkbox only -->
<label><input type=checkbox name=tac><a href="tandc.html">I agree to the terms and
conditions</a></label>
```

The usability and accessibility benefits can be maintained by placing such elements outside the `<label>` element:

```
<!-- good example - link outside label means checkbox activation area includes the checkbox
and all the label text -->
<label><input type=checkbox name=tac>I agree to the terms and conditions</label>
(read <a href="tandc.html">Terms and Conditions</a>)
```

EXAMPLE 461

The following example shows three form controls each with a label, two of which have small text showing the right format for users to use.

```
<p><label>Full name: <input name=fn> <small>Format: First Last</small></label></p>
<p><label>Age: <input name=age type=number min=0></label></p>
<p><label>Post code: <input name=pc> <small>Format: AB12 3CD</small></label></p>
```

This definition is non-normative. Implementation requirements are given below this definition.

`label` . `control`

Returns the form control that is associated with this element.

The `htmlFor` IDL attribute must [reflect](#) the `for` content attribute.

The `control` IDL attribute must return the `<label>` element's [labeled control](#), if any, or null if there isn't one.



This definition is non-normative. Implementation requirements are given below this definition.

`control` . `labels`

Returns a `NodeList` of all the `<label>` elements that the form control is associated with.

[Labelable elements](#) have a `NodeList` object associated with them that represents the list of `<label>` elements, in [tree order](#), whose [labeled control](#) is the element in question. The `labels` IDL attribute of [labelable elements](#), on getting, must return that `NodeList` object.

§ 4.10.5. The `input` element**Categories:**

[Flow content](#).

[Phrasing content](#).

If the `type` attribute is *not* in the [Hidden](#) state: [interactive content](#).

If the `type` attribute is *not* in the [Hidden](#) state: [listed](#), [labelable](#), [submittable](#), [resettable](#), and [reassociateable form-associated element](#).

If the `type` attribute is in the [Hidden](#) state: [listed](#), [submittable](#), [resettable](#), and [reassociateable form-associated element](#).

If the `type` attribute is *not* in the [Hidden](#) state: [Palpable content](#).

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Nothing.](#)

Tag omission in text/html:

No [end tag](#)

Content attributes:

[Global attributes](#)

[accept](#) - Hint for expected file type in [file upload controls](#)

[alt](#) - Replacement text for use when images are not available

[autocomplete](#) - Hint for form autofill feature

[autofocus](#) - Automatically focus the form control when the page is loaded

[checked](#) - Whether the command or control is checked

[dirname](#) - Name of form field to use for sending the element's [directionality](#) in [§4.10.22 Form submission](#)

[disabled](#) - Whether the form control is disabled

[form](#) - Associates the control with a [<form>](#) element

[formaction](#) - [URL](#) to use for [§4.10.22 Form submission](#)

[formenctype](#) - Form data set encoding type to use for [§4.10.22 Form submission](#)

[formmethod](#) - HTTP method to use for [§4.10.22 Form submission](#)

[formnovalidate](#) - Bypass form control validation for [§4.10.22 Form submission](#)

[formtarget](#) - [browsing context](#) for [§4.10.22 Form submission](#)

[height](#) - Vertical dimension

[inputmode](#) - Hint for selecting an input modality

[list](#) - List of autocomplete options

[max](#) - Maximum value

[maxlength](#) - Maximum length of value

[min](#) - Minimum value

[minlength](#) - Minimum length of value

[multiple](#) - Whether to allow multiple values

[name](#) - Name of form control to use for [§4.10.22 Form submission](#) and in the [form.elements](#) API

[pattern](#) - Pattern to be matched by the form control's value

[placeholder](#) - User-visible label to be placed within the form control

[readonly](#) - Whether to allow the value to be edited by the user

[required](#) - Whether the control is required for [§4.10.22 Form submission](#)

[size](#) - Size of the control

[src](#) - Address of the resource

[step](#) - Granularity to be matched by the form control's value

[type](#) - Type of form control

[value](#) - Value of the form control

[width](#) - Horizontal dimension

Also, the [title](#) attribute has special semantics on this element when used in conjunction with the [pattern](#) attribute.

Allowed [ARIA role attribute](#) values:

Depends upon [state of the type attribute](#).

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any [aria-*](#) attributes [applicable to the allowed roles](#).

[DOM interface](#):

```

interface HTMLInputElement : HTMLElement {
    attribute DOMString accept;
    attribute DOMString alt;
    attribute DOMString autocomplete;
    attribute boolean autofocus;
    attribute boolean defaultChecked;
    attribute boolean checked;
    attribute DOMString dirName;
    attribute boolean disabled;
    readonly attribute HTMLFormElement? form;
    readonly attribute FileList? files;
    attribute DOMString formAction;
    attribute DOMString formEnctype;
    attribute DOMString formMethod;
    attribute boolean formNoValidate;
    attribute DOMString formTarget;
    attribute unsigned long height;
    attribute boolean indeterminate;
    attribute DOMString inputMode;
    readonly attribute HTMLElement? list;
    attribute DOMString max;
    attribute long maxLength;
    attribute DOMString min;
    attribute long minLength;
    attribute boolean multiple;
    attribute DOMString name;
    attribute DOMString pattern;
    attribute DOMString placeholder;
    attribute boolean readOnly;
    attribute boolean required;
    attribute unsigned long size;
    attribute DOMString src;
    attribute DOMString step;
    attribute DOMString type;
    attribute DOMString defaultValue;
    [TreatNullAs=EmptyString] attribute DOMString value;
    attribute object? valueAsDate;
    attribute unrestricted double valueAsNumber;
    attribute unsigned long width;

    void stepUp(optional long n = 1);
    void stepDown(optional long n = 1);

    readonly attribute boolean willValidate;
    readonly attribute ValidityState validity;
    readonly attribute DOMString validationMessage;
    boolean checkValidity();
    boolean reportValidity();
    void setCustomValidity(DOMString error);

    [SameObject] readonly attribute NodeList labels;

    void select();
    attribute unsigned long selectionStart;
    attribute unsigned long selectionEnd;
    attribute DOMString selectionDirection;
    void setRangeText(DOMString replacement);
    void setRangeText(DOMString replacement, unsigned long start, unsigned long end,
optional SelectionMode selectionMode = "preserve");
    void setSelectionRange(unsigned long start, unsigned long end, optional DOMString
direction);
};

```

The `<input>` element [represents](#) a typed data field, usually with a form control to allow the user to edit the data.

The **type** attribute controls the data type of the element. It is an [enumerated attribute](#). The data type is used to select the control to use for the `<input>`. Some data types allow either a text field or combo box control to be used, based on the absence or presence of a `list` attribute on the element. The following table lists the keywords and states for the attribute — the keywords in the left column map to the state, data type and control(s) in the cells on the same row.

Keyword	State	Data type	Control type
hidden	Hidden	An arbitrary string	n/a
text	Text	Text with no line breaks	A text field or combo box
search	Search	Text with no line breaks	Search field or combo box
tel	Telephone	Text with no line breaks	A text field or combo box
url	URL	An absolute URL	A text field or combo box
email	E-mail	An e-mail address or list of e-mail addresses	A text field or combo box
password	Password	Text with no line breaks (sensitive information)	A text field that obscures data entry
date	Date	A date (year, month, day) with no time zone	A date control
month	Month	A date consisting of a year and a month with no time zone	A month control
week	Week	A date consisting of a week-year number and a week number with no time zone	A week control
time	Time	A time (hour, minute, seconds, fractional seconds) with no time zone	A time control
datetime-local	Local Date and Time	A local date and time (year, month, day, hour, minute, second, fractional seconds) with no time-zone offset information	A local date and time control
number	Number	A numerical value	A text field or combo box or spinner control
range	Range	A numerical value, with the extra semantic that the exact value is not important	A slider control or similar
color	Color	An sRGB color with 8-bit red, green, and blue components	A color well
checkbox	Checkbox	A set of zero or more values from a predefined list	A checkbox
radio	Radio Button	An enumerated value	A radio button
file	File Upload	Zero or more files each with a MIME type and optionally a file name	A label and a button
submit	submit button	An enumerated value, with the extra semantic that it must be the last value selected and initiates form submission	A button
image	image button	A coordinate, relative to a particular image's size, with the extra semantic that it must be the last value selected and initiates form submission	Either a clickable image, or a button
reset	reset button	n/a	A button
button	Button	n/a	A button

The *missing value default* is the [Text](#) state.

Which of the [accept](#), [alt](#), [autocomplete](#), [checked](#), [dirname](#), [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), [formtarget](#), [height](#), [inputmode](#), [list](#), [max](#), [maxlength](#), [min](#), [minlength](#), [multiple](#), [pattern](#), [placeholder](#), [readonly](#), [required](#), [size](#), [src](#), [step](#), and [width](#) content attributes, the [checked](#), [files](#), [valueAsDate](#), [valueAsNumber](#), and [list](#) IDL attributes, the [select\(\)](#) method, the [selectionStart](#), [selectionEnd](#), and [selectionDirection](#), IDL

attributes, the [setRangeText\(\)](#) and [setSelectionRange\(\)](#) methods, the [stepUp\(\)](#) and [stepDown\(\)](#) methods, and the input and change events **apply** to an [<input>](#) element depends on the state of its [type](#) attribute. The subsections that define each type also clearly define in normative "bookkeeping" sections which of these feature apply, and which **do not apply**, to each type. The behavior of these features depends on whether they apply or not, as defined in their various sections (q.v. for [Content attributes](#), for [APIs](#), for [events](#)).

The following table is non-normative and summarizes which of those content attributes, IDL attributes, methods, and events **apply** to each state:

	Hidden	Text , Search	URL , Telephone	E-mail	Password	Date , Month , Week , Time , Local Date and Time	Number	Range	Color	Checkbox , Radio Button	File Upload	submit button	input button
Content attributes													
accept	.	.	.	.	.	.	.	.	.	.	Yes	.	.
alt	.	.	.	.	.	.	.	.	.	.	.	.	.
autocomplete	.	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	.	.	.	.
checked	.	.	.	.	.	.	.	.	.	Yes	.	.	.
dirname	.	Yes	.	.	.	.	.	.	.	.	.	.	.
formaction	.	.	.	.	.	.	.	.	.	.	.	Yes	.
formenctype	.	.	.	.	.	.	.	.	.	.	.	Yes	.
formmethod	.	.	.	.	.	.	.	.	.	.	.	Yes	.
formnovalidate	.	.	.	.	.	.	.	.	.	.	.	Yes	.
formtarget	.	.	.	.	.	.	.	.	.	.	.	Yes	.
height	.	.	.	.	.	.	.	.	.	.	.	.	.
inputmode	.	Yes	.	.	Yes	.	.	.	.	.	.	.	.
list	.	Yes	Yes	Yes	.	Yes	Yes	Yes	Yes	.	.	.	.
max	.	.	.	.	.	Yes	Yes	Yes	.	.	.	.	.
maxlength	.	Yes	Yes	Yes	Yes	.	.	.	.	.	.	.	.
min	.	.	.	.	.	Yes	Yes	Yes	.	.	.	.	.
minlength	.	Yes	Yes	Yes	Yes	.	.	.	.	.	.	.	.
multiple	.	.	.	Yes	.	.	.	.	.	.	Yes	.	.
pattern	.	Yes	Yes	Yes	Yes	.	.	.	.	.	.	.	.
placeholder	.	Yes	Yes	Yes	Yes	.	Yes	.	.	.	.	.	.
readonly	.	Yes	Yes	Yes	Yes	Yes	Yes	.	.	.	.	.	.
required	.	Yes	Yes	Yes	Yes	Yes	Yes	.	.	Yes	Yes	.	.
size	.	Yes	Yes	Yes	Yes	.	.	.	.	.	.	.	.
src	.	.	.	.	.	.	.	.	.	.	.	.	.
step	.	.	.	.	.	Yes	Yes	Yes	.	.	.	.	.
width	.	.	.	.	.	.	.	.	.	.	.	.	.
IDL attributes and methods													

checked	.	.	.	.	.	.	.	.	.	Yes	.	.
files	.	.	.	.	.	.	.	.	.	.	Yes	.
value	default	value	value	value	value	value	value	value	value	default/on	filename	default
valueAsDate	.	.	.	.	.	Yes	.	.	.	.	.	.
valueAsNumber	.	.	.	.	.	Yes	Yes	Yes	.	.	.	.
list	.	Yes	Yes	Yes	.	Yes	Yes	Yes	Yes	.	.	.
select()	.	Yes	Yes†	Yes	Yes†	Yes†	Yes†	.	Yes†	.	Yes†	.
selectionStart	.	Yes	Yes	.	Yes	.	.	.	.	.	.	.
selectionEnd	.	Yes	Yes	.	Yes	.	.	.	.	.	.	.
selectionDirection	.	Yes	Yes	.	Yes	.	.	.	.	.	.	.
setRangeText()	.	Yes	Yes	.	Yes	.	.	.	.	.	.	.
setSelectionRange()	.	Yes	Yes	.	Yes	.	.	.	.	.	.	.
stepDown()	.	.	.	.	.	Yes	Yes	Yes	.	.	.	.
stepUp()	.	.	.	.	.	Yes	Yes	Yes	.	.	.	.
Events												
input event	.	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	.
change event	.	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	.

† If the control has no text field, the [select\(\)](#) method results in a no-op, with no "InvalidStateError" DOMException.

Some states of the [type](#) attribute define a **value sanitization algorithm**.

Each [<input>](#) element has a [value](#), which is exposed by the [value](#) IDL attribute. Some states define an **algorithm to convert a string to a number**, an **algorithm to convert a number to a string**, an **algorithm to convert a string to a [Date](#) object**, and an **algorithm to convert a [Date](#) object to a string**, which are used by [max](#), [min](#), [step](#), [valueAsDate](#), [valueAsNumber](#), [stepDown\(\)](#), and [stepUp\(\)](#).

Each [<input>](#) element has a boolean **dirty value flag**. The [dirty value flag](#) must be initially set to false when the element is created, and must be set to true whenever the user interacts with the control in a way that changes the [value](#). (It is also set to true when the value is programmatically changed, as described in the definition of the [value](#) IDL attribute.)

The [value](#) content attribute gives the default [value](#) of the [<input>](#) element. When the [value](#) content attribute is added, set, or removed, if the control's [dirty value flag](#) is false, the user agent must set the [value](#) of the element to the value of the [value](#) content attribute, if there is one, or the empty string otherwise, and then run the current [value sanitization algorithm](#), if one is defined.

Each [<input>](#) element has a [checkedness](#), which is exposed by the [checked](#) IDL attribute.

Each [<input>](#) element has a boolean **dirty checkedness flag**. When it is true, the element is said to have a **dirty checkedness**. The [dirty checkedness flag](#) must be initially set to false when the element is created, and must be set to true whenever the user interacts with the control in a way that changes the [checkedness](#).

The [checked](#) content attribute is a [boolean attribute](#) that gives the default [checkedness](#) of the [<input>](#) element. When the [checked](#) content attribute is added, if the control does not have *dirty checkedness*, the user agent must set the [checkedness](#) of the element to true; when the [checked](#) content attribute is removed, if the control does not have *dirty checkedness*, the user agent must set the [checkedness](#) of the element to false.

The [reset algorithm](#) for [<input>](#) elements is to set the [dirty value flag](#) and [dirty checkedness flag](#) back to false, set the [value](#) of the element to the value of the [value](#) content attribute, if there is one, or the empty string otherwise, set the [checkedness](#) of the element to true if the element has a [checked](#) content attribute and false if it does not, empty the list of [selected files](#), and then invoke the [value sanitization algorithm](#), if the [type](#) attribute's current state defines one.

Each [<input>](#) element can be *mutable*. Except where otherwise specified, an [<input>](#) element is always *mutable*. Similarly, except where otherwise specified, the user agent should not allow the user to modify the element's [value](#) or [checkedness](#).

When an [<input>](#) element is disabled, it is not *mutable*.

NOTE:

The [readonly](#) attribute can also in some cases (e.g., for the [Date](#) state, but not the [Checkbox](#) state) stop an [<input>](#) element from being mutable.

The [cloning steps](#) for [<input>](#) elements must propagate the [value](#), [dirty value flag](#), [checkedness](#), and [dirty checkedness flag](#) from the node being cloned to the copy.



When an [<input>](#) element is first created, the element's rendering and behavior must be set to the rendering and behavior defined for the [type](#) attribute's state, and the [value sanitization algorithm](#), if one is defined for the [type](#) attribute's state, must be invoked.

When an [<input>](#) element's [type](#) attribute changes state, the user agent must run the following steps:

1. If the previous state of the element's [type](#) attribute put the [value](#) IDL attribute in the *value* mode, and the element's [value](#) is not the empty string, and the new state of the element's [type](#) attribute puts the [value](#) IDL attribute in either the *default* mode or the *default/on* mode, then set the element's [value](#) content attribute to the element's [value](#).
2. Otherwise, if the previous state of the element's [type](#) attribute put the [value](#) IDL attribute in any mode other than the *value* mode, and the new state of the element's [type](#) attribute puts the [value](#) IDL attribute in the *value* mode, then set the [value](#) of the element to the value of the [value](#) content attribute, if there is one, or the empty string otherwise, and then set the control's [dirty value flag](#) to false.
3. Otherwise, if the previous state of the element's [type](#) attribute put the [value](#) IDL attribute in any mode other than the *filename* mode, and the new state of the element's [type](#) attribute puts the [value](#) IDL attribute in the *filename* mode, then set the [value](#) of the element to the empty string.
4. Update the element's rendering and behavior to the new state's.
5. **Signal a type change** for the element. (The [Radio Button](#) state uses this, in particular.)
6. Invoke the [value sanitization algorithm](#), if one is defined for the [type](#) attribute's new state.



The [name](#) attribute represents the element's name. The [dirname](#) attribute controls how the element's [directionality](#) is submitted. The [disabled](#) attribute is used to make the control non-interactive and to prevent its value from being submitted. The [form](#) attribute is used to explicitly associate the [<input>](#) element with its [form owner](#). The [autofocus](#) attribute controls focus. The [inputmode](#) attribute controls the user interface's input modality for the control. The [autocomplete](#) attribute controls how the user agent provides autofill behavior.

The [indeterminate](#) IDL attribute must initially be set to false. On getting, it must return the last value it was set to. On setting, it must be set to the new value. It has no effect except for changing the appearance of [checkbox](#) controls.

The [accept](#), [alt](#), [max](#), [min](#), [multiple](#), [pattern](#), [placeholder](#), [required](#), [size](#), [src](#), and [step](#) IDL attributes must [reflect](#) the respective content attributes of the same name. The [dirName](#) IDL attribute must [reflect](#) the [dirname](#) content attribute. The [readOnly](#) IDL attribute must [reflect](#) the [readonly](#) content attribute. The [defaultChecked](#) IDL attribute must [reflect](#) the [checked](#) content attribute. The [defaultValue](#) IDL attribute must [reflect](#) the [value](#) content attribute.

The [type](#) IDL attribute must [reflect](#) the respective content attribute of the same name, [limited to only known values](#). The [inputMode](#) IDL attribute must [reflect](#) the [inputmode](#) content attribute, [limited to only known values](#). The [maxLength](#) IDL attribute must [reflect](#) the [maxlength](#) content attribute, [limited to only non-negative numbers](#). The [minLength](#) IDL attribute must [reflect](#) the [minlength](#) content attribute, [limited to only non-negative numbers](#).

The IDL attributes [width](#) and [height](#) must return the rendered width and height of the image, in CSS pixels, if an image is [being rendered](#), and is being rendered to a visual medium; or else the [intrinsic width and height](#) of the image, in CSS pixels, if an image is *available* but not being rendered to a visual medium; or else 0, if no image is *available*. When the [<input>](#) element's [type](#) attribute is not in the [image button](#) state, then no image is *available*. [\[CSS-2015\]](#)

On setting, they must act as if they [reflected](#) the respective content attributes of the same name.

The [willValidate](#), [validity](#), and [validationMessage](#) IDL attributes, and the [checkValidity\(\)](#), [reportValidity\(\)](#), and [setCustomValidity\(\)](#) methods, are part of the [constraint validation API](#). The [labels](#) IDL attribute provides a list of the element's `<label>`s. The [select\(\)](#), [selectionStart](#), [selectionEnd](#), [selectionDirection](#), [setRangeText\(\)](#), and [setSelectionRange\(\)](#) methods and IDL attributes expose the element's text selection. The [autofocus](#), [disabled](#), [form](#), and [name](#) IDL attributes are part of the element's forms API.

§ 4.10.5.1. States of the `type` attribute

§ 4.10.5.1.1. HIDDEN STATE (`type=hidden`)

Allowed [ARIA role attribute](#) values:

None

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

When an `<input>` element's `type` attribute is in the [Hidden](#) state, the rules in this section apply.

The `<input>` element [represents](#) a value that is not intended to be examined or manipulated by the user.

Constraint validation: If an `<input>` element's `type` attribute is in the [Hidden](#) state, it is [barred from constraint validation](#).

If the `name` attribute is present and has a value that is a [case-sensitive](#) match for the string `"_charset_"`, then the element's value attribute must be omitted.

Bookkeeping details

- The [value](#) IDL attribute [applies](#) to this element and is in mode [default](#).
- The following content attributes must not be specified and [do not apply](#) to the element: [accept](#), [alt](#), [autocomplete](#), [checked](#), [dirname](#), [formation](#), [formenctype](#), [formmethod](#), [formnovalidate](#), [formtarget](#), [height](#), [inputmode](#), [list](#), [max](#), [maxlength](#), [min](#), [minlength](#), [multiple](#), [pattern](#), [placeholder](#), [readonly](#), [required](#), [size](#), [src](#), [step](#), and [width](#).
- The following IDL attributes and methods [do not apply](#) to the element: [checked](#), [files](#), [list](#), [selectionStart](#), [selectionEnd](#), [selectionDirection](#), [valueAsDate](#), and [valueAsNumber](#) IDL attributes; [select\(\)](#), [setRangeText\(\)](#), [setSelectionRange\(\)](#), [stepDown\(\)](#), and [stepUp\(\)](#) methods.
- The input and change events [do not apply](#).

§ 4.10.5.1.2. TEXT (`type=text`) STATE AND SEARCH STATE (`type=search`)

Allowed [ARIA role attribute](#) values:

`'textbox'` (default - [do not set](#)) or `'combobox'`.

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any `aria-*` attributes [applicable to the allowed roles](#).

When an `<input>` element's `type` attribute is in the [Text](#) state or the [Search](#) state, the rules in this section apply.

The `<input>` element [represents](#) a one line plain text edit control for the element's [value](#).

NOTE:

The difference between the [Text](#) state and the [Search](#) state is primarily stylistic: on platforms where search fields are distinguished from regular text fields, the [Search](#) state might result in an appearance consistent with the platform's search fields rather than appearing like a regular text field.

If the element is *mutable*, its [value](#) should be editable by the user. User agents must not allow users to insert U+000A LINE

FEED (LF) or U+000D CARRIAGE RETURN (CR) characters into the element's [value](#).

If the element is *mutable*, the user agent should allow the user to change the writing direction of the element, setting it either to a left-to-right writing direction or a right-to-left writing direction. If the user does so, the user agent must then run the following steps:

1. Set the element's [dir](#) attribute to ["ltr"](#) if the user selected a left-to-right writing direction, and ["rtl"](#) if the user selected a right-to-left writing direction.
2. [Queue a task](#) to [fire a simple event](#) that bubbles named input at the [<input>](#) element.

The value attribute, if specified, must have a value that contains no U+000A LINE FEED (LF) or U+000D CARRIAGE RETURN (CR) characters.

The [value sanitization algorithm](#) is as follows: [Strip line breaks](#) from the [value](#).

Bookkeeping details

- The following common [<input>](#) element content attributes, IDL attributes, and methods [apply](#) to the element: [autocomplete](#), [dirname](#), [inputmode](#), [list](#), [maxlength](#), [minlength](#), [pattern](#), [placeholder](#), [readonly](#), [required](#), and [size](#) content attributes; [list](#), [selectionStart](#), [selectionEnd](#), [selectionDirection](#), and [value](#) IDL attributes; [select\(\)](#), [setRangeText\(\)](#), and [setSelectionRange\(\)](#) methods.
- The [value](#) IDL attribute is in mode [value](#).
- The input and change events [apply](#).
- The following content attributes must not be specified and [do not apply](#) to the element: [accept](#), [alt](#), [checked](#), [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), [formtarget](#), [height](#), [max](#), [min](#), [multiple](#), [src](#), [step](#), and [width](#).
- The following IDL attributes and methods [do not apply](#) to the element: [checked](#), [files](#), [valueAsDate](#), and [valueAsNumber](#) IDL attributes; [stepDown\(\)](#) and [stepUp\(\)](#) methods.

§ 4.10.5.1.3. TELEPHONE STATE ([type=tel](#))

Allowed [ARIA role attribute](#) values:

[‘textbox’](#) (default - [do not set](#)) or [‘combobox’](#).

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any [aria-* attributes](#) [applicable to the allowed roles](#).

When an [<input>](#) element's [type](#) attribute is in the [Telephone](#) state, the rules in this section apply.

The [<input>](#) element [represents](#) a control for editing a telephone number given in the element's [value](#).

If the element is *mutable*, its [value](#) should be editable by the user. User agents may change the spacing and, with care, the punctuation of [values](#) that the user enters. User agents must not allow users to insert U+000A LINE FEED (LF) or U+000D CARRIAGE RETURN (CR) characters into the element's [value](#).

The value attribute, if specified, must have a value that contains no U+000A LINE FEED (LF) or U+000D CARRIAGE RETURN (CR) characters.

The [value sanitization algorithm](#) is as follows: [Strip line breaks](#) from the [value](#).

NOTE:

Unlike the [URL](#) and [E-mail](#) types, the [Telephone](#) type does not enforce a particular syntax. This is intentional; in practice, telephone number fields tend to be free-form fields, because there are a wide variety of valid phone numbers.

Systems that need to enforce a particular format are encouraged to use the [pattern](#) attribute or the [setCustomValidity\(\)](#) method to hook into the client-side validation mechanism.

Bookkeeping details

- The following common [<input>](#) element content attributes, IDL attributes, and methods [apply](#) to the element: [autocomplete](#), [list](#), [maxlength](#), [minlength](#), [pattern](#), [placeholder](#), [readonly](#), [required](#), and [size](#) content attributes; [list](#), [selectionStart](#), [selectionEnd](#), [selectionDirection](#), and [value](#) IDL attributes; [select\(\)](#), [setRangeText\(\)](#), and [setSelectionRange\(\)](#) methods.

- The [value](#) IDL attribute is in mode [value](#).
- The input and change events [apply](#).
- The following content attributes must not be specified and [do not apply](#) to the element: [accept](#), [alt](#), [checked](#), [dirname](#), [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), [formtarget](#), [height](#), [inputmode](#), [max](#), [min](#), [multiple](#), [src](#), [step](#), and [width](#).
- The following IDL attributes and methods [do not apply](#) to the element: [checked](#), [files](#), [valueAsDate](#), and [valueAsNumber](#) IDL attributes; [stepDown\(\)](#) and [stepUp\(\)](#) methods.

§ 4.10.5.1.4. URL STATE (`type=url`)

Allowed [ARIA role attribute](#) values:

[‘textbox’](#) (default - [do not set](#)) or [‘combobox’](#).

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any [aria-* attributes](#) [applicable to the allowed roles](#).

When an `<input>` element’s [type](#) attribute is in the [URL](#) state, the rules in this section apply.

The `<input>` element [represents](#) a control for editing a single [absolute URL](#) given in the element’s [value](#).

If the element is *mutable*, the user agent should allow the user to change the URL represented by its [value](#). User agents may allow the user to set the [value](#) to a string that is not a [valid absolute URL](#), but may also or instead automatically escape characters entered by the user so that the [value](#) is always a [valid absolute URL](#) (even if that isn’t the actual value seen and edited by the user in the interface). User agents should allow the user to set the [value](#) to the empty string. User agents must not allow users to insert U+000A LINE FEED (LF) or U+000D CARRIAGE RETURN (CR) characters into the [value](#).

The [value](#) attribute, if specified and not empty, must have a value that is a [valid URL potentially surrounded by spaces](#) that is also an [absolute URL](#).

The [value sanitization algorithm](#) is as follows: [Strip line breaks](#) from the [value](#), then [strip leading and trailing whitespace](#) from the [value](#).

Constraint validation: While the [value](#) of the element is neither the empty string nor a [valid absolute URL](#), the element is [suffering from a type mismatch](#).

Bookkeeping details

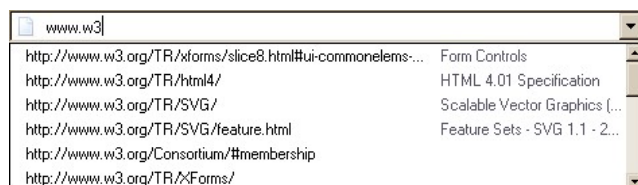
- The following common `<input>` element content attributes, IDL attributes, and methods [apply](#) to the element: [autocomplete](#), [list](#), [maxlength](#), [minlength](#), [pattern](#), [placeholder](#), [readonly](#), [required](#), and [size](#) content attributes; [list](#), [selectionStart](#), [selectionEnd](#), [selectionDirection](#), and [value](#) IDL attributes; [select\(\)](#), [setRangeText\(\)](#), and [setSelectionRange\(\)](#) methods.
- The [value](#) IDL attribute is in mode [value](#).
- The input and change events [apply](#).
- The following content attributes must not be specified and [do not apply](#) to the element: [accept](#), [alt](#), [checked](#), [dirname](#), [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), [formtarget](#), [height](#), [inputmode](#), [max](#), [min](#), [multiple](#), [src](#), [step](#), and [width](#).
- The following IDL attributes and methods [do not apply](#) to the element: [checked](#), [files](#), [valueAsDate](#), and [valueAsNumber](#) IDL attributes; [stepDown\(\)](#) and [stepUp\(\)](#) methods.

EXAMPLE 462

If a document contained the following markup:

```
<input type="url" name="location" list="urls">
<datalist id="urls">
  <option label="MIME: Format of Internet Message Bodies" value="https://tools.ietf.org/html/rfc2045">
  <option label="HTML 4.01 Specification" value="https://www.w3.org/TR/html4/">
  <option label="Form Controls" value="https://www.w3.org/TR/xforms/slice8.html#ui-commonelems-hint">
  <option label="Scalable Vector Graphics (SVG) 1.1 Specification" value="https://www.w3.org/TR/SVG/">
  <option label="Feature Sets - SVG 1.1 - 20030114" value="https://www.w3.org/TR/SVG/feature.html">
  <option label="The Single UNIX Specification, Version 3" value="https://www.unix-systems.org/version3/">
</datalist>
```

...and the user had typed "www.w3", and the user agent had also found that the user had visited <https://www.w3.org/Consortium/membership> and <https://www.w3.org/TR/XForms/> in the recent past, then the rendering might look like this:



The first four URLs in this sample consist of the four URLs in the author-specified list that match the text the user has entered, sorted in some user agent-defined manner (maybe by how frequently the user refers to those URLs). Note how the user agent is using the knowledge that the values are URLs to allow the user to omit the scheme part and perform intelligent matching on the domain name.

The last two URLs (and probably many more, given the scrollbar's indications of more values being available) are the matches from the user agent's session history data. This data is not made available to the page DOM. In this particular case, the user agent has no titles to provide for those values.

§ 4.10.5.1.5. E-MAIL STATE (`type=email`)

Allowed [ARIA role attribute](#) values:

`'textbox'` (default - [do not set](#)) or `'combobox'`.

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

When an `<input>` element's `type` attribute is in the [E-mail](#) state, the rules in this section apply.

How the [E-mail](#) state operates depends on whether the `multiple` attribute is specified or not.

➤ When the `multiple` attribute is not specified on the element

The `<input>` element [represents](#) a control for editing an e-mail address given in the element's `value`.

If the element is *mutable*, the user agent should allow the user to change the e-mail address represented by its `value`. User agents may allow the user to set the `value` to a string that is not a [valid e-mail address](#). The user agent should act in a manner consistent with expecting the user to provide a single e-mail address. User agents should al-

low the user to set the [value](#) to the empty string. User agents must not allow users to insert U+000A LINE FEED (LF) or U+000D CARRIAGE RETURN (CR) characters into the [value](#). User agents may transform the [value](#) for display and editing; in particular, user agents should convert punycode in the domain labels of the [value](#) to IDN in the display and vice versa.

Constraint validation: While the user interface is representing input that the user agent cannot convert to punycode, the control is [suffering from bad input](#).

The [value](#) attribute, if specified and not empty, must have a value that is a single [valid e-mail address](#).

The [value sanitization algorithm](#) is as follows: [Strip line breaks](#) from the [value](#), then [strip leading and trailing whitespace](#) from the [value](#).

Constraint validation: While the [value](#) of the element is neither the empty string nor a single [valid e-mail address](#), the element is [suffering from a type mismatch](#).

↪ When the [multiple](#) attribute is specified on the element

The `<input>` element [represents](#) a control for adding, removing, and editing the e-mail addresses given in the element's [values](#).

If the element is *mutable*, the user agent should allow the user to add, remove, and edit the e-mail addresses represented by its [values](#). User agents may allow the user to set any individual value in the list of [values](#) to a string that is not a [valid e-mail address](#), but must not allow users to set any individual value to a string containing U+002C COMMA (,), U+000A LINE FEED (LF), or U+000D CARRIAGE RETURN (CR) characters. User agents should allow the user to remove all the addresses in the element's [values](#). User agents may transform the [values](#) for display and editing; in particular, user agents should convert punycode in the domain labels of the [value](#) to IDN in the display and vice versa.

Constraint validation: While the user interface describes a situation where an individual value contains a U+002C COMMA (,) or is representing input that the user agent cannot convert to punycode, the control is [suffering from bad input](#).

Whenever the user changes the element's [values](#), the user agent must run the following steps:

1. Let *latest values* be a copy of the element's [values](#).
2. [Strip leading and trailing whitespace](#) from each value in *latest values*.
3. Let the element's [value](#) be the result of concatenating all the values in *latest values*, separating each value from the next by a single U+002C COMMA character (,), maintaining the list's order.

The [value](#) attribute, if specified, must have a value that is a [valid e-mail address list](#).

The [value sanitization algorithm](#) is as follows:

1. [Split on commas](#) the element's [value](#), [strip leading and trailing whitespace](#) from each resulting token, if any, and let the element's [values](#) be the (possibly empty) resulting list of (possibly empty) tokens, maintaining the original order.
2. Let the element's [value](#) be the result of concatenating the element's [values](#), separating each value from the next by a single U+002C COMMA character (,), maintaining the list's order.

Constraint validation: While the [value](#) of the element is not a [valid e-mail address list](#), the element is [suffering from a type mismatch](#).

When the [multiple](#) attribute is set or removed, the user agent must run the [value sanitization algorithm](#).

A **valid e-mail address** is a string that matches the `email` production of the following ABNF, the character set for which is Unicode. This ABNF implements the extensions described in RFC 1123. [\[ABNF\]](#) [\[RFC5322\]](#) [\[RFC1034\]](#) [\[RFC1123\]](#)

```

email      = 1*( atext / "." ) "@" label *( "." label )
label      = let-dig [ [ ldh-str ] let-dig ] ; limited to a length of 63 characters by RFC 1034 section 3.5
atext      = < as defined in RFC 5322 section 3.2.3 >
let-dig    = < as defined in RFC 1034 section 3.5 >
ldh-str    = < as defined in RFC 1034 section 3.5 >

```

NOTE:

This requirement is a [willful violation](#) of RFC 5322, which defines a syntax for e-mail addresses that is simultaneously too strict (before the "@" character), too vague (after the "@" character), and too lax (allowing comments, whitespace characters, and quoted strings in manners unfamiliar to most users) to be of practical use here.

The following JavaScript- and Perl-compatible regular expression is an implementation of the above definition.

```

/^ [a-zA-Z0-9.!#$%&'*\+\/=?^_`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?(?:\.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)*$/

```

A **valid e-mail address list** is a [set of comma-separated tokens](#), where each token is itself a [valid e-mail address](#). To obtain the list of tokens from a [valid e-mail address list](#), an implementation must [split the string on commas](#).

Bookkeeping details

- The following common [<input>](#) element content attributes, IDL attributes, and methods [apply](#) to the element: [autocomplete](#), [list](#), [maxlength](#), [minlength](#), [multiple](#), [pattern](#), [placeholder](#), [readonly](#), [required](#), and [size](#) content attributes; [list](#) and [value](#) IDL attributes; [select\(\)](#) method.
- The [value](#) IDL attribute is in mode [value](#).
- The input and change events [apply](#).
- The following content attributes must not be specified and [do not apply](#) to the element: [accept](#), [alt](#), [checked](#), [dirname](#), [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), [formtarget](#), [height](#), [inputmode](#), [max](#), [min](#), [src](#), [step](#), and [width](#).
- The following IDL attributes and methods [do not apply](#) to the element: [checked](#), [files](#), [selectionStart](#), [selectionEnd](#), [selectionDirection](#), [valueAsDate](#), and [valueAsNumber](#) IDL attributes; [select\(\)](#), [setRangeText\(\)](#), [setSelectionRange\(\)](#), [stepDown\(\)](#) and [stepUp\(\)](#) methods.

§ 4.10.5.1.6. PASSWORD STATE (type=password)

Allowed ARIA role attribute values:

[‘textbox’](#) (default - [do not set](#)).

Allowed ARIA state and property attributes:

Global aria-* attributes

Any aria-* attributes [applicable to the allowed roles](#).

When an [<input>](#) element’s [type](#) attribute is in the [Password](#) state, the rules in this section apply.

The [<input>](#) element [represents](#) a one line plain text edit control for the element’s [value](#). The user agent should obscure the value so that people other than the user cannot see it.

If the element is *mutable*, its [value](#) should be editable by the user. User agents must not allow users to insert U+000A LINE FEED (LF) or U+000D CARRIAGE RETURN (CR) characters into the [value](#).

The [value](#) attribute, if specified, must have a value that contains no U+000A LINE FEED (LF) or U+000D CARRIAGE RETURN (CR) characters.

The **value sanitization algorithm** is as follows: [Strip line breaks](#) from the [value](#).

Bookkeeping details

- The following common [<input>](#) element content attributes, IDL attributes, and methods [apply](#) to the element: [autocomplete](#), [inputmode](#), [maxlength](#), [minlength](#), [pattern](#), [placeholder](#), [readonly](#), [required](#), and [size](#) content attributes; [selectionStart](#), [selectionEnd](#),

[selectionDirection](#), and [value](#) IDL attributes; [select\(\)](#), [setRangeText\(\)](#), and [setSelectionRange\(\)](#) methods.

- The [value](#) IDL attribute is in mode [value](#).
- The input and change events [apply](#).
- The following content attributes must not be specified and [do not apply](#) to the element: [accept](#), [alt](#), [checked](#), [dirname](#), [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), [formtarget](#), [height](#), [list](#), [max](#), [min](#), [multiple](#), [src](#), [step](#), and [width](#).
- The following IDL attributes and methods [do not apply](#) to the element: [checked](#), [files](#), [list](#), [valueAsDate](#), and [valueAsNumber](#) IDL attributes; [stepDown\(\)](#) and [stepUp\(\)](#) methods.

§ 4.10.5.1.7. DATE STATE ([type=date](#))

Allowed [ARIA role attribute](#) values:

None

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

When an [<input>](#) element's [type](#) attribute is in the [Date](#) state, the rules in this section apply.

The [<input>](#) element [represents](#) a control for setting the element's [value](#) to a string representing a specific [date](#).

NOTE:

[date](#) values represent a "floating" time and do not include [time zone](#) information. Care is needed when converting values of this type to or from date data types in JavaScript and other programming languages. In many cases, an implicit time-of-day and time zone are used to create a global ("incremental") time (an integer value that represents the offset from some arbitrary epoch time). Processing or conversion of these values, particularly across time zones, can change the value of the date itself. [\[TIMEZONE\]](#)

If the element is *mutable*, the user agent should allow the user to change the [date](#) represented by its [value](#), as obtained by [parsing a date](#) from it. User agents must not allow the user to set the [value](#) to a non-empty string that is not a [valid date string](#). If the user agent provides a user interface for selecting a [date](#), then the [value](#) must be set to a [valid date string](#) representing the user's selection. User agents should allow the user to set the [value](#) to the empty string.

Constraint validation: While the user interface describes input that the user agent cannot convert to a [valid date string](#), the control is [suffering from bad input](#).

NOTE:

See [§4.10.1.8 Date, time, and number formats](#) for a discussion of the difference between the input format and submission format for date, time, and number form controls, and the [implementation notes](#) regarding localization of form controls.

The [value](#) attribute, if specified and not empty, must have a value that is a [valid date string](#).

The [value sanitization algorithm](#) is as follows: If the [value](#) of the element is not a [valid date string](#), then set it to the empty string instead.

The [min](#) attribute, if specified, must have a value that is a [valid date string](#). The [max](#) attribute, if specified, must have a value that is a [valid date string](#).

The [step](#) attribute is expressed in days. The [step scale factor](#) is 86,400,000 (which converts the days to milliseconds, which is the base unit of comparison for the conversion algorithms below). The [default step](#) is 1 day.

When the element is [suffering from a step mismatch](#), the user agent may round the element's [value](#) to the nearest [date](#) for which the element would not [suffer from a step mismatch](#).

The [algorithm to convert a string to a number](#), given a string *input*, is as follows: If [parsing a date](#) from *input* results in an error, then return an error; otherwise, return the number of milliseconds elapsed from midnight UTC on the morning of 1970-01-01 (the time represented by the value "1970-01-01T00:00:00.0Z") to midnight UTC on the morning of the parsed [date](#), ignoring leap seconds.

The **algorithm to convert a number to a string**, given a number *input*, is as follows: Return a [valid date string](#) that represents the [date](#) that, in UTC, is current *input* milliseconds after midnight UTC on the morning of 1970-01-01 (the time represented by the value "1970-01-01T00:00:00.0Z").

The **algorithm to convert a string to a Date object**, given a string *input*, is as follows: If [parsing a date](#) from *input* results in an error, then return an error; otherwise, return a [new Date object](#) representing midnight UTC on the morning of the parsed [date](#).

The **algorithm to convert a Date object to a string**, given a [Date object](#) *input*, is as follows: Return a [valid date string](#) that represents the [date](#) current at the time represented by *input* in the UTC time zone.

Bookkeeping details

- The following common [<input>](#) element content attributes, IDL attributes, and methods [apply](#) to the element: [autocomplete](#), [list](#), [max](#), [min](#), [readonly](#), [required](#), and [step](#) content attributes; [list](#), [value](#), [valueAsDate](#), and [valueAsNumber](#) IDL attributes; [select\(\)](#), [stepDown\(\)](#), and [stepUp\(\)](#) methods.
- The [value](#) IDL attribute is in mode [value](#).
- The input and change events [apply](#).
- The following content attributes must not be specified and [do not apply](#) to the element: [accept](#), [alt](#), [checked](#), [dirname](#), [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), [formtarget](#), [height](#), [inputmode](#), [maxlength](#), [minlength](#), [multiple](#), [pattern](#), [placeholder](#), [size](#), [src](#), and [width](#).
- The following IDL attributes and methods [do not apply](#) to the element: [checked](#), [selectionStart](#), [selectionEnd](#), and [selectionDirection](#) IDL attributes; [setRangeText\(\)](#), and [setSelectionRange\(\)](#) methods.

§ 4.10.5.1.8. MONTH STATE (type=month)

Allowed [ARIA role attribute](#) values:

None

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

When an [<input>](#) element's [type](#) attribute is in the [Month](#) state, the rules in this section apply.

The [<input>](#) element [represents](#) a control for setting the element's [value](#) to a string representing a specific [month](#).

If the element is *mutable*, the user agent should allow the user to change the [month](#) represented by its [value](#), as obtained by [parsing a month](#) from it. User agents must not allow the user to set the [value](#) to a non-empty string that is not a [valid month string](#). If the user agent provides a user interface for selecting a [month](#), then the [value](#) must be set to a [valid month string](#) representing the user's selection. User agents should allow the user to set the [value](#) to the empty string.

Constraint validation: While the user interface describes input that the user agent cannot convert to a [valid month string](#), the control is [suffering from bad input](#).

NOTE:

See [§4.10.1.8 Date, time, and number formats](#) for a discussion of the difference between the input format and submission format for date, time, and number form controls, and the [implementation notes](#) regarding localization of form controls.

The [value](#) attribute, if specified and not empty, must have a value that is a [valid month string](#).

The **value sanitization algorithm** is as follows: If the [value](#) of the element is not a [valid month string](#), then set it to the empty string instead.

The [min](#) attribute, if specified, must have a value that is a [valid month string](#). The [max](#) attribute, if specified, must have a value that is a [valid month string](#).

The [step](#) attribute is expressed in months. The [step scale factor](#) is 1 (units of whole months are the base unit of comparison for the conversion algorithms below). The [default step](#) is 1 month.

When the element is [suffering from a step mismatch](#), the user agent may round the element's [value](#) to the nearest [month](#) for

which the element would not [suffer from a step mismatch](#).

The algorithm to convert a string to a number, given a string *input*, is as follows: If [parsing a month](#) from *input* results in an error, then return an error; otherwise, return the number of months between January 1970 and the parsed [month](#).

The algorithm to convert a number to a string, given a number *input*, is as follows: Return a [valid month string](#) that represents the [month](#) that has *input* months between it and January 1970.

The algorithm to convert a string to a Date object, given a string *input*, is as follows: If [parsing a month](#) from *input* results in an error, then return an error; otherwise, return a [new Date object](#) representing midnight UTC on the morning of the first day of the parsed [month](#).

The algorithm to convert a Date object to a string, given a [Date](#) object *input*, is as follows: Return a [valid month string](#) that represents the [month](#) current at the time represented by *input* in the UTC time zone.

Bookkeeping details

- The following common `<input>` element content attributes, IDL attributes, and methods [apply](#) to the element: [autocomplete](#), [list](#), [max](#), [min](#), [readonly](#), [required](#), and [step](#) content attributes; [list](#), [value](#), [valueAsDate](#), and [valueAsNumber](#) IDL attributes; [select\(\)](#), [stepDown\(\)](#), and [stepUp\(\)](#) methods.
- The [value](#) IDL attribute is in mode [value](#).
- The input and change events [apply](#).
- The following content attributes must not be specified and [do not apply](#) to the element: [accept](#), [alt](#), [checked](#), [dirname](#), [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), [formtarget](#), [height](#), [inputmode](#), [maxlength](#), [minlength](#), [multiple](#), [pattern](#), [placeholder](#), [size](#), [src](#), and [width](#).
- The following IDL attributes and methods [do not apply](#) to the element: [checked](#), [files](#), [selectionStart](#), [selectionEnd](#), and [selectionDirection](#) IDL attributes; [setRangeText\(\)](#), and [setSelectionRange\(\)](#) methods.

§ 4.10.5.1.9. WEEK STATE (`type=week`)

Allowed [ARIA role attribute](#) values:

None

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

When an `<input>` element's [type](#) attribute is in the [Week](#) state, the rules in this section apply.

The `<input>` element [represents](#) a control for setting the element's [value](#) to a string representing a specific [week](#) beginning on a Monday, at midnight UTC.

If the element is *mutable*, the user agent should allow the user to change the [week](#) represented by its [value](#), as obtained by [parsing a week](#) from it. User agents must not allow the user to set the [value](#) to a non-empty string that is not a [valid week string](#). If the user agent provides a user interface for selecting a [week](#), then the [value](#) must be set to a [valid week string](#) representing the user's selection. User agents should allow the user to set the [value](#) to the empty string.

Constraint validation: While the user interface describes input that the user agent cannot convert to a [valid week string](#), the control is [suffering from bad input](#).

NOTE:

See [§4.10.1.8 Date, time, and number formats](#) for a discussion of the difference between the input format and submission format for date, time, and number form controls, and the [implementation notes](#) regarding localization of form controls.

The [value](#) attribute, if specified and not empty, must have a value that is a [valid week string](#).

The [value sanitization algorithm](#) is as follows: If the [value](#) of the element is not a [valid week string](#), then set it to the empty string instead.

The [min](#) attribute, if specified, must have a value that is a [valid week string](#). The [max](#) attribute, if specified, must have a value that is a [valid week string](#).

The [step](#) attribute is expressed in weeks. The [step scale factor](#) is 604,800,000 (which converts the weeks to milliseconds, which is the base unit of comparison for the conversion algorithms below). The [default step](#) is 1 week. The [default step base](#) is -259,200,000 (the start of week 1970-W01 which is the Monday 3 days before 1970-01-01).

When the element is [suffering from a step mismatch](#), the user agent may round the element's [value](#) to the nearest [week](#) for which the element would not [suffer from a step mismatch](#).

The algorithm to convert a string to a number, given a string *input*, is as follows: If [parsing a week string](#) from *input* results in an error, then return an error; otherwise, return the number of milliseconds elapsed from midnight UTC on the morning of 1970-01-01 (the time represented by the value "1970-01-01T00:00:00.0Z") to midnight UTC on the morning of the Monday of the parsed [week](#), ignoring leap seconds.

The algorithm to convert a number to a string, given a number *input*, is as follows: Return a [valid week string](#) that represents the [week](#) that, in UTC, is current *input* milliseconds after midnight UTC on the morning of 1970-01-01 (the time represented by the value "1970-01-01T00:00:00.0Z").

The algorithm to convert a string to a Date object, given a string *input*, is as follows: If [parsing a week](#) from *input* results in an error, then return an error; otherwise, return a [new Date object](#) representing midnight UTC on the morning of the Monday of the parsed [week](#).

The algorithm to convert a Date object to a string, given a Date object *input*, is as follows: Return a [valid week string](#) that represents the [week](#) current at the time represented by *input* in the UTC time zone.

Bookkeeping details

- The following common [<input>](#) element content attributes, IDL attributes, and methods [apply](#) to the element: [autocomplete](#), [list](#), [max](#), [min](#), [readonly](#), [required](#), and [step](#) content attributes; [list](#), [value](#), [valueAsDate](#), and [valueAsNumber](#) IDL attributes; [select\(\)](#), [stepDown\(\)](#), and [stepUp\(\)](#) methods.
- The [value](#) IDL attribute is in mode [value](#).
- The input and change events [apply](#).
- The following content attributes must not be specified and [do not apply](#) to the element: [accept](#), [alt](#), [checked](#), [dirname](#), [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), [formtarget](#), [height](#), [inputmode](#), [maxlength](#), [minlength](#), [multiple](#), [pattern](#), [placeholder](#), [size](#), [src](#), and [width](#).
- The following IDL attributes and methods [do not apply](#) to the element: [checked](#), [files](#), [selectionStart](#), [selectionEnd](#), and [selectionDirection](#) IDL attributes; [setRangeText\(\)](#), and [setSelectionRange\(\)](#) methods.

§ 4.10.5.1.10. TIME STATE (type=time)

Allowed [ARIA role attribute](#) values:

None

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

When an [<input>](#) element's [type](#) attribute is in the [Time](#) state, the rules in this section apply.

The [<input>](#) element [represents](#) a control for setting the element's [value](#) to a string representing a specific [time](#).

If the element is *mutable*, the user agent should allow the user to change the [time](#) represented by its [value](#), as obtained by [parsing a time](#) from it. User agents must not allow the user to set the [value](#) to a non-empty string that is not a [valid time string](#). If the user agent provides a user interface for selecting a [time](#), then the [value](#) must be set to a [valid time string](#) representing the user's selection. User agents should allow the user to set the [value](#) to the empty string.

Constraint validation: While the user interface describes input that the user agent cannot convert to a [valid time string](#), the control is [suffering from bad input](#).

NOTE:

See [§4.10.1.8 Date, time, and number formats](#) for a discussion of the difference between the input format and submission format for date, time, and number form controls, and the [implementation notes](#) regarding localization of form controls.

The `value` attribute, if specified and not empty, must have a value that is a [valid time string](#).

The value sanitization algorithm is as follows: If the `value` of the element is not a [valid time string](#), then set it to the empty string instead.

The form control [has a periodic domain](#).

The `min` attribute, if specified, must have a value that is a [valid time string](#). The `max` attribute, if specified, must have a value that is a [valid time string](#).

The `step` attribute is expressed in seconds. The [step scale factor](#) is 1000 (which converts the seconds to milliseconds, which is the base unit of comparison for the conversion algorithms below). The [default step](#) is 60 seconds.

When the element is [suffering from a step mismatch](#), the user agent may round the element's `value` to the nearest [time](#) for which the element would not [suffer from a step mismatch](#).

The algorithm to convert a string to a number, given a string `input`, is as follows: If [parsing a time](#) from `input` results in an error, then return an error; otherwise, return the number of milliseconds elapsed from midnight to the parsed [time](#) on a day with no time changes.

The algorithm to convert a number to a string, given a number `input`, is as follows: Return a [valid time string](#) that represents the [time](#) that is `input` milliseconds after midnight on a day with no time changes.

The algorithm to convert a string to a Date object, given a string `input`, is as follows: If [parsing a time](#) from `input` results in an error, then return an error; otherwise, return a [new Date object](#) representing the parsed [time](#) in UTC on 1970-01-01.

The algorithm to convert a Date object to a string, given a Date object `input`, is as follows: Return a [valid time string](#) that represents the UTC [time](#) component that is represented by `input`.

Bookkeeping details

- The following common `<input>` element content attributes, IDL attributes, and methods [apply](#) to the element: [autocomplete](#), [list](#), [max](#), [min](#), [readonly](#), [required](#), and [step](#) content attributes; [list](#), [value](#), [valueAsDate](#), and [valueAsNumber](#) IDL attributes; [select\(\)](#), [stepDown\(\)](#), and [stepUp\(\)](#) methods.
- The `value` IDL attribute is in mode [value](#).
- The input and change events [apply](#).
- The following content attributes must not be specified and [do not apply](#) to the element: [accept](#), [alt](#), [checked](#), [dirname](#), [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), [formtarget](#), [height](#), [inputmode](#), [maxlength](#), [minlength](#), [multiple](#), [pattern](#), [placeholder](#), [size](#), [src](#), and [width](#).
- The following IDL attributes and methods [do not apply](#) to the element: [checked](#), [files](#), [selectionStart](#), [selectionEnd](#), and [selectionDirection](#) IDL attributes; [setRangeText\(\)](#), and [setSelectionRange\(\)](#) methods.

§ 4.10.5.1.11. LOCAL DATE AND TIME STATE (type=datetime-local)

When an `<input>` element's `type` attribute is in the [Local Date and Time](#) state, the rules in this section apply.

The `<input>` element [represents](#) a control for setting the element's `value` to a string representing a [local date and time](#), with no time-zone offset information.

If the element is [mutable](#) and the user agent provides a user interface for selecting a [local date and time](#), then the `value` must be set to a [valid normalized global date and time string](#) representing the user's selection. User agents should allow the user to set the `value` to the empty string.

Constraint validation: While the user interface describes input that the user agent cannot convert to a [valid normalized global date and time string](#), the control is [suffering from bad input](#).

NOTE:

See [§4.10.1.8 Date, time, and number formats](#) for a discussion of the difference between the input format and submission format for date, time, and number form controls, and the [implementation notes](#) regarding localization of form controls.

The `value` attribute, if specified and not empty, must have a value that is a [valid floating date and time string](#).

The **value sanitization algorithm** is as follows: If the [value](#) of the element is a [valid floating date and time string](#), then set it to a [valid normalized floating date and time string](#) representing the same date and time; otherwise, set it to the empty string instead.

The [min](#) attribute, if specified, must have a value that is a [valid floating date and time string](#). The [max](#) attribute, if specified, must have a value that is a [valid floating date and time string](#).

The [step](#) attribute is expressed in seconds. The [step scale factor](#) is 1000 (which converts the seconds to milliseconds, which is the base unit of comparison for the conversion algorithms below). The [default step](#) is 60 seconds.

When the element is [suffering from a step mismatch](#), the user agent may round the element's [value](#) to the nearest [floating date and time](#) for which the element would not [suffer from a step mismatch](#).

The **algorithm to convert a string to a number**, given a string *input*, is as follows: If [parsing a date and time](#) from *input* results in an error, then return an error; otherwise, return the number of milliseconds elapsed from midnight on the morning of 1970-01-01 (the time represented by the value "1970-01-01T00:00:00.0") to the parsed [floating date and time](#), ignoring leap seconds.

The **algorithm to convert a number to a string**, given a number *input*, is as follows: Return a [valid normalized floating date and time string](#) that represents the date and time that is *input* milliseconds after midnight on the morning of 1970-01-01 (the time represented by the value "1970-01-01T00:00:00.0").

Bookkeeping details

- The following common [<input>](#) element content attributes, IDL attributes, and methods [apply](#) to the element: [autocomplete](#), [list](#), [max](#), [min](#), [readonly](#), [required](#), and [step](#) content attributes; [list](#), [value](#), and [valueAsNumber](#) IDL attributes; [select\(\)](#), [stepDown\(\)](#), and [stepUp\(\)](#) methods.
- The [value](#) IDL attribute is in mode [value](#).
- The input and change events [apply](#).
- The following content attributes must not be specified and [do not apply](#) to the element: [accept](#), [alt](#), [checked](#), [dirname](#), [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), [formtarget](#), [height](#), [inputmode](#), [maxlength](#), [minlength](#), [multiple](#), [pattern](#), [placeholder](#), [size](#), [src](#), and [width](#).
- The following IDL attributes and methods [do not apply](#) to the element: [checked](#), [files](#), [selectionStart](#), [selectionEnd](#), [selectionDirection](#), and [valueAsDate](#) IDL attributes; [setRangeText\(\)](#), and [setSelectionRange\(\)](#) methods.

EXAMPLE 463

The following example shows part of a flight booking application. The application uses an [<input>](#) element with its [type](#) attribute set to `datetime-local`, and it then interprets the given date and time in the time zone of the selected airport.

```
<fieldset>
  <legend>Destination</legend>
  <p><label>Airport: <input type=text name=to list=airports></label></p>
  <p><label>Departure time: <input type=datetime-local name=totime step=3600></label></p>
</fieldset>
<datalist id=airports>
  <option value=ATL label="Atlanta">
  <option value=MEM label="Memphis">
  <option value=LHR label="London Heathrow">
  <option value=LAX label="Los Angeles">
  <option value=FRA label="Frankfurt">
</datalist>
```

Allowed [ARIA role attribute](#) values:

[‘spinbutton’](#) (default - do not set).

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

When an `<input>` element’s [type](#) attribute is in the [Number](#) state, the rules in this section apply.

The `<input>` element [represents](#) a control for setting the element’s [value](#) to a string representing a number.

If the element is *mutable*, the user agent should allow the user to change the number represented by its [value](#), as obtained from applying the [rules for parsing floating-point number values](#) to it. User agents must not allow the user to set the [value](#) to a non-empty string that is not a [valid floating-point number](#). If the user agent provides a user interface for selecting a number, then the [value](#) must be set to the [best representation of the number representing the user’s selection as a floating-point number](#). User agents should allow the user to set the [value](#) to the empty string.

Constraint validation: While the user interface describes input that the user agent cannot convert to a [valid floating-point number](#), the control is [suffering from bad input](#).

This specification does not define what user interface user agents are to use; user agent vendors are encouraged to consider what would best serve their users’ needs. For example, a user agent in Persian or Arabic markets might support Persian and Arabic numeric input (converting it to the format required for submission as described above). Similarly, a user agent designed for the French market might display the value with apostrophes between thousands and commas before the decimals, and allow the user to enter a value in that manner, internally converting it to the submission format described above.

See [§4.10.1.8 Date, time, and number formats](#) for a discussion of the difference between the input format and submission format for date, time, and number form controls, and the [implementation notes](#) regarding localization of form controls.

The [value](#) attribute, if specified and not empty, must have a value that is a [valid floating-point number](#).

The value sanitization algorithm is as follows: If the [value](#) of the element is not a [valid floating-point number](#), then set it to the empty string instead.

The [min](#) attribute, if specified, must have a value that is a [valid floating-point number](#). The [max](#) attribute, if specified, must have a value that is a [valid floating-point number](#).

The [step scale factor](#) is 1. The [default step](#) is 1 (allowing only integers to be selected by the user, unless the [step base](#) has a non-integer value).

When the element is [suffering from a step mismatch](#), the user agent may round the element’s [value](#) to the nearest number for which the element would not [suffer from a step mismatch](#). If there are two such numbers, user agents are encouraged to pick the one nearest positive infinity.

The algorithm to convert a string to a number, given a string *input*, is as follows: If applying the [rules for parsing floating-point number values](#) to *input* results in an error, then return an error; otherwise, return the resulting number.

The algorithm to convert a number to a string, given a number *input*, is as follows: Return a [valid floating-point number](#) that represents *input*.

Bookkeeping details

- The following common `<input>` element content attributes, IDL attributes, and methods [apply](#) to the element: [autocomplete](#), [list](#), [max](#), [min](#), [placeholder](#), [readonly](#), [required](#), and [step](#) content attributes; [list](#), [value](#), and [valueAsNumber](#) IDL attributes; [select\(\)](#), [stepDown\(\)](#), and [stepUp\(\)](#) methods.
- The [value](#) IDL attribute is in mode [value](#).
- The input and change events [apply](#).
- The following content attributes must not be specified and do not apply to the element: [accept](#), [alt](#), [checked](#), [dirname](#), [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), [formtarget](#), [height](#), [inputmode](#), [maxlength](#), [minlength](#), [multiple](#), [pattern](#), [size](#), [src](#), and [width](#).

- The following IDL attributes and methods **do not apply** to the element: [checked](#), [files](#), [selectionStart](#), [selectionEnd](#), [selectionDirection](#), and [valueAsDate](#) IDL attributes; [setRangeText\(\)](#), and [setSelectionRange\(\)](#) methods.

EXAMPLE 464

Here is an example of using a numeric input control:

```
<label>How much do you want to charge? $<input type=number min=0 step=0.01 name=price></label>
```

As described above, a user agent might support numeric input in the user's local format, converting it to the format required for submission as described above. This might include handling grouping separators (as in "872,000,000,000") and various decimal separators (such as "3,99" vs "3.99") or using local digits (such as those in Arabic, Devanagari, Persian, and Thai).

NOTE:

The `type=number` state is not appropriate for input that happens to only consist of numbers but isn't strictly speaking a number. For example, it would be inappropriate for credit card numbers or US postal codes. A simple way of determining whether to use `type=number` is to consider whether it would make sense for the input control to have a spinbox interface (e.g., with "up" and "down" arrows). Getting a credit card number wrong by 1 in the last digit isn't a minor mistake, it's as wrong as getting every digit incorrect. So it would not make sense for the user to select a credit card number using "up" and "down" buttons. When a spinbox interface is not appropriate, `type=text` is probably the right choice (possibly with a `pattern` attribute).

§ 4.10.5.1.13. RANGE STATE (`type=range`)

Allowed ARIA role attribute values:

`'slider'` (default - [do not set](#)).

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any `aria-*` attributes [applicable to the allowed roles](#).

When an `<input>` element's `type` attribute is in the [Range](#) state, the rules in this section apply.

The `<input>` element [represents](#) a control for setting the element's `value` to a string representing a number, but with the caveat that the exact value is not important, letting user agents provide a simpler interface than they do for the [Number](#) state.

If the element is *mutable*, the user agent should allow the user to change the number represented by its `value`, as obtained from applying the [rules for parsing floating-point number values](#) to it. User agents must not allow the user to set the `value` to a string that is not a [valid floating-point number](#). If the user agent provides a user interface for selecting a number, then the `value` must be set to a [best representation of the number representing the user's selection as a floating-point number](#). User agents must not allow the user to set the `value` to the empty string.

Constraint validation: While the user interface describes input that the user agent cannot convert to a [valid floating-point number](#), the control is [suffering from bad input](#).

The `value` attribute, if specified, must have a value that is a [valid floating-point number](#).

The value sanitization algorithm is as follows: If the `value` of the element is not a [valid floating-point number](#), then set it to the [best representation, as a floating-point number](#), of the [default value](#).

The **default value** is the [minimum](#) plus half the difference between the [minimum](#) and the [maximum](#), unless the [maximum](#) is less than the [minimum](#), in which case the [default value](#) is the [minimum](#).

When the element is [suffering from an underflow](#), the user agent must set the element's `value` to the [best representation, as a floating-point number](#), of the [minimum](#).

When the element is [suffering from an overflow](#), if the [maximum](#) is not less than the [minimum](#), the user agent must set the

element's [value](#) to a [valid floating-point number](#) that represents the [maximum](#).

When the element is [suffering from a step mismatch](#), the user agent must round the element's [value](#) to the nearest number for which the element would not [suffer from a step mismatch](#), and which is greater than or equal to the [minimum](#), and, if the [maximum](#) is not less than the [minimum](#), which is less than or equal to the [maximum](#), if there is a number that matches these constraints. If two numbers match these constraints, then user agents must use the one nearest to positive infinity.

EXAMPLE 465

For example, the markup `<input type="range" min=0 max=100 step=20 value=50>` results in a range control whose initial value is 60.

EXAMPLE 466

Here is an example of a range control using an autocomplete list with the [list](#) attribute. This could be useful if there are values along the full range of the control that are especially important, such as preconfigured light levels or typical speed limits in a range control used as a speed control. The following markup fragment:

```
<input type="range" min="-100" max="100" value="0" step="10" name="power" list="powers">
<datalist id="powers">
  <option value="0">
  <option value="-30">
  <option value="30">
  <option value="++50">
</datalist>
```

...with the following style sheet applied:

```
input { height: 75px; width: 49px; background: #D5CCBB; color: black; }
```

...might render as:



Note how the user agent determined the orientation of the control from the ratio of the style-sheet-specified height and width properties. The colors were similarly derived from the style sheet. The tick marks, however, were derived from the markup. In particular, the [step](#) attribute has not affected the placement of tick marks, the user agent deciding to only use the author-specified completion values and then adding longer tick marks at the extremes.

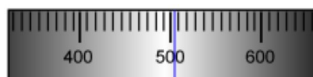
Note also how the invalid value `++50` was completely ignored.

EXAMPLE 467

For another example, consider the following markup fragment:

```
<input name=x type=range min=100 max=700 step=9.09090909 value=509.090909>
```

A user agent could display in a variety of ways, for instance:



Or, alternatively, for instance:



The user agent could pick which one to display based on the dimensions given in the style sheet. This would allow it to maintain the same resolution for the tick marks, despite the differences in width.

EXAMPLE 468

Finally, here is an example of a range control with two labeled values:

```
<input type="range" name="a" list="a-values">
<datalist id="a-values">
  <option value="10" label="Low">
  <option value="90" label="High">
</datalist>
```

With styles that make the control draw vertically, it might look as follows:

**NOTE:**

In this state, the range and step constraints are enforced even during user input, and there is no way to set the value to the empty string.

The [min](#) attribute, if specified, must have a value that is a [valid floating-point number](#). The [default minimum](#) is 0. The [max](#) attribute, if specified, must have a value that is a [valid floating-point number](#). The [default maximum](#) is 100.

The [step scale factor](#) is 1. The [default step](#) is 1 (allowing only integers, unless the [min](#) attribute has a non-integer value).

The algorithm to convert a string to a number, given a string *input*, is as follows: If applying the [rules for parsing floating-point number values](#) to *input* results in an error, then return an error; otherwise, return the resulting number.

The algorithm to convert a number to a string, given a number *input*, is as follows: Return the [best representation, as a floating-point number](#), of *input*.

Bookkeeping details

The following common `<input>` element content attributes, IDL attributes, and methods [apply](#) to the element: [autocomplete](#), [list](#), [max](#), [min](#), and [step](#) content attributes; [list](#), [value](#), and [valueAsNumber](#) IDL attributes; [stepDown\(\)](#) and [stepUp\(\)](#) methods.

- The [value](#) IDL attribute is in mode [value](#).
- The input and change events [apply](#).

- The following content attributes must not be specified and **do not apply** to the element: [accept](#), [alt](#), [checked](#), [dirname](#), [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), [formtarget](#), [height](#), [inputmode](#), [maxlength](#), [minlength](#), [multiple](#), [pattern](#), [placeholder](#), [readonly](#), [required](#), [size](#), [src](#), and [width](#).
- The following IDL attributes and methods **do not apply** to the element: [checked](#), [files](#), [selectionStart](#), [selectionEnd](#), [selectionDirection](#), and [valueAsDate](#) IDL attributes; [select\(\)](#), [setRangeText\(\)](#), and [setSelectionRange\(\)](#) methods.

§ 4.10.5.1.14. COLOR STATE (`type=color`)

Allowed [ARIA role attribute](#) values:

None

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

When an `<input>` element's `type` attribute is in the [Color](#) state, the rules in this section apply.

The `<input>` element [represents](#) a color well control, for setting the element's [value](#) to a string representing a [simple color](#).

NOTE:

In this state, there is always a color picked, and there is no way to set the value to the empty string.

If the element is *mutable*, the user agent should allow the user to change the color represented by its [value](#), as obtained from applying the [rules for parsing simple color values](#) to it. User agents must not allow the user to set the [value](#) to a string that is not a [valid lowercase simple color](#). If the user agent provides a user interface for selecting a color, then the [value](#) must be set to the result of using the [rules for serializing simple color values](#) to the user's selection. User agents must not allow the user to set the [value](#) to the empty string.

Constraint validation: While the user interface describes input that the user agent cannot convert to a [valid lowercase simple color](#), the control is [suffering from bad input](#).

The [value](#) attribute, if specified and not empty, must have a value that is a [valid simple color](#).

The value sanitization algorithm is as follows: If the [value](#) of the element is a [valid simple color](#), then set it to the [value](#) of the element [converted to ASCII lowercase](#); otherwise, set it to the string `"#000000"`.

Bookkeeping details

- The following common `<input>` element content attributes and IDL attributes [apply](#) to the element: [autocomplete](#) and [list](#) content attributes; [list](#) and [value](#) IDL attributes; [select\(\)](#) method.
- The [value](#) IDL attribute is in mode [value](#).
- The input and change events [apply](#).
- The following content attributes must not be specified and **do not apply** to the element: [accept](#), [alt](#), [checked](#), [dirname](#), [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), [formtarget](#), [height](#), [inputmode](#), [max](#), [maxlength](#), [min](#), [minlength](#), [multiple](#), [pattern](#), [placeholder](#), [readonly](#), [required](#), [size](#), [src](#), [step](#), and [width](#).
- The following IDL attributes and methods **do not apply** to the element: [checked](#), [files](#), [selectionStart](#), [selectionEnd](#), [selectionDirection](#), [valueAsDate](#), and [valueAsNumber](#) IDL attributes; [setRangeText\(\)](#), [setSelectionRange\(\)](#), [stepDown\(\)](#), and [stepUp\(\)](#) methods.

§ 4.10.5.1.15. CHECKBOX STATE (`type=checkbox`)

Allowed [ARIA role attribute](#) values:

`'checkbox'` (default - [do not set](#)) or `'menuitemcheckbox'`.

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

When an `<input>` element's `type` attribute is in the [Checkbox](#) state, the rules in this section apply.

The `<input>` element [represents](#) a two-state control that represents the element's [checkedness](#) state. If the element's [checkedness](#) state is true, the control represents a positive selection, and if it is false, a negative selection. If the element's indeterminate IDL attribute is set to true, then the control's selection should be obscured as if the control was in a third, indeterminate, state.

NOTE:

The control is never a true tri-state control, even if the element's indeterminate IDL attribute is set to true. The indeterminate IDL attribute only gives the appearance of a third state.

If the element is *mutable*, then: The [pre-click activation steps](#) consist of setting the element's [checkedness](#) to its opposite value (i.e., true if it is false, false if it is true), and of setting the element's indeterminate IDL attribute to false. The [canceled activation steps](#) consist of setting the [checkedness](#) and the element's indeterminate IDL attribute back to the values they had before the [pre-click activation steps](#) were run. The [activation behavior](#) is to [fire a simple event](#) that bubbles named `input` at the element and then [fire a simple event](#) that bubbles named `change` at the element.

If the element is not *mutable*, it has no [activation behavior](#).

Constraint validation: If the element is *required* and its [checkedness](#) is false, then the element is [suffering from being missing](#).

This definition is non-normative. Implementation requirements are given below this definition.

`input . indeterminate [ = value ]`

When set, overrides the rendering of [checkbox](#) controls so that the current value is not visible.

Bookkeeping details

- The following common `<input>` element content attributes and IDL attributes [apply](#) to the element: [checked](#), and [required](#) content attributes; [checked](#) and [value](#) IDL attributes.
- The [value](#) IDL attribute is in mode [default/on](#).
- The `input` and `change` events [apply](#).
- The following content attributes must not be specified and [do not apply](#) to the element: [accept](#), [alt](#), [autocomplete](#), [dirname](#), [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), [formtarget](#), [height](#), [inputmode](#), [list](#), [max](#), [maxlength](#), [min](#), [minlength](#), [multiple](#), [pattern](#), [placeholder](#), [readonly](#), [size](#), [src](#), [step](#), and [width](#).
- The following IDL attributes and methods [do not apply](#) to the element: [files](#), [list](#), [selectionStart](#), [selectionEnd](#), [selectionDirection](#), [valueAsDate](#), and [valueAsNumber](#) IDL attributes; [select\(\)](#), [setRangeText\(\)](#), [setSelectionRange\(\)](#), [stepDown\(\)](#), and [stepUp\(\)](#) methods.

§ 4.10.5.1.16. RADIO BUTTON STATE (type=radio)

Allowed **ARIA role attribute** values:

`'radio'` (default - [do not set](#)) or `'menuitemradio'`.

Allowed **ARIA state and property attributes**:

[Global aria-* attributes](#)

Any `aria-*` attributes [applicable to the allowed roles](#).

When an `<input>` element's [type](#) attribute is in the [Radio Button](#) state, the rules in this section apply.

The `<input>` element [represents](#) a control that, when used in conjunction with other `<input>` elements, forms a *radio button group* in which only one control can have its [checkedness](#) state set to true. If the element's [checkedness](#) state is true, the control represents the selected control in the group, and if it is false, it indicates a control in the group that is not selected.

The *radio button group* that contains an `<input>` element *a* also contains all the other `<input>` elements *b* that fulfill all of the following conditions:

- The `<input>` element *b*'s [type](#) attribute is in the [Radio Button](#) state.
- Either *a* and *b* have the same [form owner](#), or they both have no [form owner](#).

- Both a and b are in the same [home subtree](#).
- They both have a [name](#) attribute, their [name](#) attributes are not empty, and the value of a 's [name](#) attribute is a [compatibility caseless](#) match for the value of b 's [name](#) attribute.

A document must not contain an `<input>` element whose *radio button group* contains only that element.

When any of the following phenomena occur, if the element's [checkedness](#) state is true after the occurrence, the [checkedness](#) state of all the other elements in the same *radio button group* must be set to false:

- The element's [checkedness](#) state is set to true (for whatever reason).
- The element's [name](#) attribute is set, changed, or removed.
- The element's [form owner](#) changes.
- A [type change is signalled](#) for the element.

If the element R is *mutable*, then: The [pre-click activation steps](#) for R consist of getting a reference to the element in R 's [radio button group](#) that has its [checkedness](#) set to true, if any, and then setting R 's [checkedness](#) to true. The [canceled activation steps](#) for R consist of checking if the element to which a reference was obtained in the [pre-click activation steps](#), if any, is still in what is now R 's [radio button group](#), if it still has one, and if so, setting that element's [checkedness](#) to true; or else, if there was no such element, or that element is no longer in R 's [radio button group](#), or if R no longer has a [radio button group](#), setting R 's [checkedness](#) to false. The [activation behavior](#) for R is to [fire a simple event](#) that bubbles named `input` at R and then [fire a simple event](#) that bubbles named `change` at R .

If the element is not *mutable*, it has no [activation behavior](#).

Constraint validation: If an element in the *radio button group* is *required*, and all of the `<input>` elements in the *radio button group* have a [checkedness](#) that is false, then the element is [suffering from being missing](#).

NOTE:

If none of the radio buttons in a [radio button group](#) are checked when they are [inserted into the document](#), then they will all be initially unchecked in the interface, until such time as one of them is checked (either by the user or by script).

Bookkeeping details

- The following common `<input>` element content attributes and IDL attributes [apply](#) to the element: [checked](#) and [required](#) content attributes; [checked](#) and [value](#) IDL attributes.
- The [value](#) IDL attribute is in mode [default/on](#).
- The `input` and `change` events [apply](#).
- The following content attributes must not be specified and [do not apply](#) to the element: [accept](#), [alt](#), [autocomplete](#), [dirname](#), [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), [formtarget](#), [height](#), [inputmode](#), [list](#), [max](#), [maxlength](#), [min](#), [minlength](#), [multiple](#), [pattern](#), [placeholder](#), [readonly](#), [size](#), [src](#), [step](#), and [width](#).
- The following IDL attributes and methods [do not apply](#) to the element: [files](#), [list](#), [selectionStart](#), [selectionEnd](#), [selectionDirection](#), [valueAsDate](#), and [valueAsNumber](#) IDL attributes; [select\(\)](#), [setRangeText\(\)](#), [setSelectionRange\(\)](#), [stepDown\(\)](#), and [stepUp\(\)](#) methods.

§ 4.10.5.1.17. FILE UPLOAD STATE (type=file)

Allowed [ARIA role attribute](#) values:

None

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

When an `<input>` element's [type](#) attribute is in the [File Upload](#) state, the rules in this section apply.

The `<input>` element [represents](#) a list of **selected files**, each file consisting of a file name, a file type, and a file body (the contents of the file).

File names must not contain [path components](#), even in the case that a user has selected an entire directory hierarchy or multiple files with the same name from different directories. **Path components**, for the purposes of the [File Upload](#) state, are those parts of file names that are separated by U+005C REVERSE SOLIDUS character (\) characters.

Unless the `multiple` attribute is set, there must be no more than one file in the list of [selected files](#).

If the element is *mutable*, then the element's [activation behavior](#) is to run the following steps:

1. If the algorithm is not [allowed to show a popup](#), then abort these steps without doing anything else.
2. Return, but continue running these steps [in parallel](#).
3. Optionally, wait until any prior execution of this algorithm has terminated.
4. Display a prompt to the user requesting that the user specify some files. If the `multiple` attribute is not set, there must be no more than one file selected; otherwise, any number may be selected. Files can be from the filesystem or created on the fly, e.g., a picture taken from a camera connected to the user's device.
5. Wait for the user to have made their selection.
6. [Queue a task](#) to first update the element's [selected files](#) so that it represents the user's selection, then [fire a simple event](#) that bubbles named `input` at the `<input>` element, and finally [fire a simple event](#) that bubbles named `change` at the `<input>` element.

If the element is *mutable*, the user agent should allow the user to change the files on the list in other ways also, e.g., adding or removing files by drag-and-drop. When the user does so, the user agent must [queue a task](#) to first update the element's [selected files](#) so that it represents the user's new selection, then [fire a simple event](#) that bubbles named `input` at the `<input>` element, and finally [fire a simple event](#) that bubbles named `change` at the `<input>` element.

If the element is not *mutable*, it has no [activation behavior](#) and the user agent must not allow the user to change the element's selection.

Constraint validation: If the element is *required* and the list of [selected files](#) is empty, then the element is [suffering from being missing](#).



The **accept** attribute may be specified to provide user agents with a hint of what file types will be accepted.

If specified, the attribute must consist of a [set of comma-separated tokens](#), each of which must be an [ASCII case-insensitive match](#) for one of the following:

The string "audio/*"

Indicates that sound files are accepted.

The string "video/*"

Indicates that video files are accepted.

The string "image/*"

Indicates that image files are accepted.

A valid MIME type with no parameters

Indicates that files of the specified type are accepted.

A string whose first character is a U+002E FULL STOP character (.)

Indicates that files with the specified file extension are accepted.

The tokens must not be [ASCII case-insensitive matches](#) for any of the other tokens (i.e., duplicates are not allowed). To obtain the list of tokens from the attribute, the user agent must [split the attribute value on commas](#).

User agents may use the value of this attribute to display a more appropriate user interface than a generic file picker. For instance, given the value `image/*`, a user agent could offer the user the option of using a local camera or selecting a photograph from their photo collection; given the value `audio/*`, a user agent could offer the user the option of recording a clip using a headset microphone.

User agents should prevent the user from selecting files that are not accepted by one (or more) of these tokens.

NOTE:

Authors are encouraged to specify both any MIME types and any corresponding extensions when looking for data in a specific format.

EXAMPLE 469

For example, consider an application that converts Microsoft Word documents to Open Document Format files. Since Microsoft Word documents are described with a wide variety of MIME types and extensions, the site can list several, as follows:

```
<input type="file" accept=".doc,.docx,.xml,application/msword,application/vnd.openxmlformats-officedocument.wordprocessingml.document">
```

On platforms that only use file extensions to describe file types, the extensions listed here can be used to filter the allowed documents, while the MIME types can be used with the system's type registration table (mapping MIME types to extensions used by the system), if any, to determine any other extensions to allow. Similarly, on a system that does not have file names or extensions but labels documents with MIME types internally, the MIME types can be used to pick the allowed files, while the extensions can be used if the system has an extension registration table that maps known extensions to MIME types used by the system.

⚠Warning! Extensions tend to be ambiguous (e.g., there are an untold number of formats that use the ".dat" extension, and users can typically quite easily rename their files to have a ".doc" extension even if they are not Microsoft Word documents), and MIME types tend to be unreliable (e.g., many formats have no formally registered types, and many formats are in practice labeled using a number of different MIME types). Authors are reminded that, as usual, data received from a client should be treated with caution, as it may not be in an expected format even if the user is not hostile and the user agent fully obeyed the [accept](#) attribute's requirements.

EXAMPLE 470

For historical reasons, the [value](#) IDL attribute prefixes the file name with the string "C:\\fakepath\\". Some legacy user agents actually included the full path (which was a security vulnerability). As a result of this, obtaining the file name from the [value](#) IDL attribute in a backwards-compatible way is non-trivial. The following function extracts the file name in a suitably compatible manner:

```
function extractFilename(path) {
  if (path.substr(0, 12) == "C:\\fakepath\\")
    return path.substr(12); // modern browser
  var x;
  x = path.lastIndexOf('/');
  if (x >= 0) // Unix-based path
    return path.substr(x+1);
  x = path.lastIndexOf('\\');
  if (x >= 0) // Windows-based path
    return path.substr(x+1);
  return path; // just the file name
}
```

This can be used as follows:

```
<p><input type=file name=image onchange="updateFilename(this.value)"></p>
<p>The name of the file you picked is: <span id="filename">(none)</span></p>
<script>
  function updateFilename(path) {
    var name = extractFilename(path);
    document.getElementById('filename').textContent = name;
  }
</script>
```

**Bookkeeping details**

The following common [<input>](#) element content attributes and IDL attributes [apply](#) to the element: [accept](#), [multiple](#), and [required](#) content attributes; [files](#) and [value](#) IDL attributes; [select\(\)](#) method.

- The [value](#) IDL attribute is in mode [filename](#).
- The input and change events [apply](#).
- The following content attributes must not be specified and [do not apply](#) to the element: [alt](#), [autocomplete](#), [checked](#), [dirname](#), [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), [formtarget](#), [height](#), [inputmode](#), [list](#), [max](#), [maxlength](#), [min](#), [minlength](#), [pattern](#), [placeholder](#), [readonly](#), [size](#), [src](#), [step](#), and [width](#).
- The element's [value](#) attribute must be omitted.
- The following IDL attributes and methods [do not apply](#) to the element: [checked](#), [list](#), [selectionStart](#), [selectionEnd](#), [selectionDirection](#), [valueAsDate](#), and [valueAsNumber](#) IDL attributes; [setRangeText\(\)](#), [setSelectionRange\(\)](#), [stepDown\(\)](#), and [stepUp\(\)](#) methods.

§ 4.10.5.1.18. SUBMIT BUTTON STATE (type=submit)

Allowed [ARIA role attribute](#) values:

[‘button’](#) (default - [do not set](#)).

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

When an [<input>](#) element's [type](#) attribute is in the [submit button](#) state, the rules in this section apply.

The `<input>` element [represents](#) a button that, when activated, submits the form. If the element has a [value](#) attribute, the button's label must be the value of that attribute; otherwise, it must be an implementation-defined string that means "Submit" or some such. The element is a [button](#), specifically a [submit button](#). 

NOTE:

Since the default label is implementation-defined, and the width of the button typically depends on the button's label, the button's width can leak a few bits of fingerprintable information. These bits are likely to be strongly correlated to the identity of the user agent and the user's locale.

If the element is *mutable*, then the element's [activation behavior](#) is as follows: if the element has a [form owner](#), and the element's [node document](#) is [fully active](#), [submit](#) the [form owner](#) from the `<input>` element; otherwise, do nothing.

If the element is not *mutable*, it has no [activation behavior](#).

The [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), and [formtarget](#) attributes are [attributes for form submission](#).

NOTE:

The [formnovalidate](#) attribute can be used to make submit buttons that do not trigger the constraint validation.

Bookkeeping details

- The following common `<input>` element content attributes and IDL attributes [apply](#) to the element: [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), and [formtarget](#) content attributes; [value](#) IDL attribute.
- The [value](#) IDL attribute is in mode [default](#).
- The following content attributes must not be specified and [do not apply](#) to the element: [accept](#), [alt](#), [autocomplete](#), [checked](#), [dirname](#), [height](#), [inputmode](#), [list](#), [max](#), [maxlength](#), [min](#), [minlength](#), [multiple](#), [pattern](#), [placeholder](#), [readonly](#), [required](#), [size](#), [src](#), [step](#), and [width](#).
- The following IDL attributes and methods [do not apply](#) to the element: [checked](#), [files](#), [list](#), [selectionStart](#), [selectionEnd](#), [selectionDirection](#), [valueAsDate](#), and [valueAsNumber](#) IDL attributes; [select\(\)](#), [setRangeText\(\)](#), [setSelectionRange\(\)](#), [stepDown\(\)](#), and [stepUp\(\)](#) methods.
- The input and change events [do not apply](#).

§ 4.10.5.1.19. IMAGE BUTTON STATE (`type=image`)

Allowed ARIA role attribute values:

[‘button’](#) (default - [do not set](#)), [‘link’](#), [‘menuitem’](#), [‘menuitemcheckbox’](#), [‘menuitemradio’](#) or [‘radio’](#).

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

When an `<input>` element's [type](#) attribute is in the [image button](#) state, the rules in this section apply.

The `<input>` element [represents](#) either an image from which a user can select a coordinate and submit the form, or alternatively a button from which the user can submit the form. The element is a [button](#), specifically a [submit button](#).

NOTE:

The coordinate is sent to the server [during form submission](#) by sending two entries for the element, derived from the name of the control but with `.x` and `.y` appended to the name with the `x` and `y` components of the coordinate respectively.



The image is given by the [src](#) attribute. The [src](#) attribute must be present, and must contain a [valid non-empty URL potentially surrounded by spaces](#) referencing a non-interactive, optionally animated, image resource that is neither paged nor scripted.

When any of the these events occur

- the `<input>` element's `type` attribute is first set to the [Image Button](#) state (possibly when the element is first created), and the `src` attribute is present
- the `<input>` element's `type` attribute is changed back to the [Image Button](#) state, and the `src` attribute is present, and its value has changed since the last time the `type` attribute was in the [Image Button](#) state
- the `<input>` element's `type` attribute is in the [Image Button](#) state, and the `src` attribute is set or changed

then unless the user agent cannot support images, or its support for images has been disabled, or the user agent only fetches images on demand, or the `src` attribute's value is the empty string, the user agent must [parse](#) the value of the `src` attribute value, relative to the element's [node document](#), and if that is successful, run these substeps:

1. Let *request* be a new [request](#) whose [URL](#) is the [resulting URL string](#), [client](#) is the element's [node document](#)'s Window object's [environment settings object](#), `type` is "image", [destination](#) is "subresource", [omit-Origin-header flag](#) is set, [credentials mode](#) is "include", and whose [use-URL-credentials flag](#) is set.
2. [Fetch](#) *request*.

Fetching the image must [delay the load event](#) of the element's [node document](#) until the [task](#) that is [queued](#) by the [networking task source](#) once the resource has been fetched (defined below) has been run.

If the image was successfully obtained, with no network errors, and the image's type is a supported image type, and the image is a valid image of that type, then the image is said to be *available*. If this is true before the image is completely downloaded, each [task](#) that is [queued](#) by the [networking task source](#) while the image is being fetched must update the presentation of the image appropriately.

The user agent should apply the [image sniffing rules](#) to determine the type of the image, with the image's [associated Content-Type headers](#) giving the *official type*. If these rules are not applied, then the type of the image must be the type given by the image's [associated Content-Type headers](#).

User agents must not support non-image resources with the `<input>` element. User agents must not run executable code embedded in the image resource. User agents must only display the first page of a multipage resource. User agents must not allow the resource to act in an interactive fashion, but should honor any animation in the resource.

The [task](#) that is [queued](#) by the [networking task source](#) once the resource has been fetched, must, if the download was successful and the image is *available*, [queue a task to fire a simple event](#) named `load` at the `<input>` element; and otherwise, if the fetching process fails without a response from the remote server, or completes but the image is not a valid or supported image, [queue a task to fire a simple event](#) named `error` on the `<input>` element.



The `alt` attribute provides the textual label for the button for users and user agents who cannot use the image. The `alt` attribute must be present, and must contain a non-empty string giving the label that would be appropriate for an equivalent button if the image was unavailable.

The `<input>` element supports [dimension attributes](#).



If the `src` attribute is set, and the image is *available* and the user agent is configured to display that image, then: The element [represents](#) a control for selecting a [coordinate](#) from the image specified by the `src` attribute; if the element is *mutable*, the user agent should allow the user to select this [coordinate](#), and the element's [activation behavior](#) is as follows: if the element has a [form owner](#), and the element's [node document](#) is [fully active](#), take the user's selected [coordinate](#), and [submit](#) the `<input>` element's [form owner](#) from the `<input>` element. If the user activates the control without explicitly selecting a coordinate, then the coordinate (0,0) must be assumed.

Otherwise, the element [represents](#) a submit button whose label is given by the value of the `alt` attribute; if the element is *mutable*, then the element's [activation behavior](#) is as follows: if the element has a [form owner](#), and the element's [node document](#) is [fully active](#), set the [selected coordinate](#) to (0,0), and [submit](#) the `<input>` element's [form owner](#) from the `<input>` element.

ment.

In either case, if the element is *mutable* but has no [form owner](#) or the element's [node document](#) is not [fully active](#), then its [activation behavior](#) must be to do nothing. If the element is not *mutable*, it has no [activation behavior](#).

The **selected coordinate** must consist of an *x*-component and a *y*-component. The coordinates represent the position relative to the edge of the image, with the coordinate space having the positive *x* direction to the right, and the positive *y* direction downwards.

The *x*-component must be a [valid integer](#) representing a number *x* in the range $-(border_{left} + padding_{left}) \leq x \leq width + border_{right} + padding_{right}$, where *width* is the rendered width of the image, *border_{left}* is the width of the border on the left of the image, *padding_{left}* is the width of the padding on the left of the image, *border_{right}* is the width of the border on the right of the image, and *padding_{right}* is the width of the padding on the right of the image, with all dimensions given in CSS pixels.

The *y*-component must be a [valid integer](#) representing a number *y* in the range $-(border_{top} + padding_{top}) \leq y \leq height + border_{bottom} + padding_{bottom}$, where *height* is the rendered height of the image, *border_{top}* is the width of the border above the image, *padding_{top}* is the width of the padding above the image, *border_{bottom}* is the width of the border below the image, and *padding_{bottom}* is the width of the padding below the image, with all dimensions given in CSS pixels.

Where a border or padding is missing, its width is zero CSS pixels.



The [formation](#), [formenctype](#), [formmethod](#), [formnovalidate](#), and [formtarget](#) attributes are [attributes for form submission](#).

This definition is non-normative. Implementation requirements are given below this definition.

image . width [= *value*]

image . height [= *value*]

These attributes return the actual rendered dimensions of the image, or zero if the dimensions are not known.

They can be set, to change the corresponding content attributes.

Bookkeeping details

- The following common `<input>` element content attributes and IDL attributes [apply](#) to the element: [alt](#), [formation](#), [formenctype](#), [formmethod](#), [formnovalidate](#), [formtarget](#), [height](#), [src](#), and [width](#) content attributes; [value](#) IDL attribute.
- The [value](#) IDL attribute is in mode [default](#).
- The following content attributes must not be specified and [do not apply](#) to the element: [accept](#), [autocomplete](#), [checked](#), [dirname](#), [inputmode](#), [list](#), [max](#), [maxlength](#), [min](#), [minlength](#), [multiple](#), [pattern](#), [placeholder](#), [readonly](#), [required](#), [size](#), and [step](#).
- The element's [value](#) attribute must be omitted.
- The following IDL attributes and methods [do not apply](#) to the element: [checked](#), [files](#), [list](#), [selectionStart](#), [selectionEnd](#), [selectionDirection](#), [valueAsDate](#), and [valueAsNumber](#) IDL attributes; [select\(\)](#), [setRangeText\(\)](#), [setSelectionRange\(\)](#), [stepDown\(\)](#), and [stepUp\(\)](#) methods.
- The input and change events [do not apply](#).

NOTE:

Many aspects of this state's behavior are similar to the behavior of the `<img>` element. Readers are encouraged to read that section, where many of the same requirements are described in more detail.

EXAMPLE 471

Take the following form:

```
<form action="process.cgi">
  <input type=image src=map.png name=where alt="Show location list">
</form>
```

If the user clicked on the image at coordinate (127,40) then the URL used to submit the form would be "process.cgi?where.x=127&where.y=40".

(In this example, it's assumed that for users who don't see the map, and who instead just see a button labeled "Show location list", clicking the button will cause the server to show a list of locations to pick from instead of the map.)

§ 4.10.5.1.20. RESET BUTTON STATE (type=reset)

Allowed [ARIA role attribute](#) values:

[‘button’](#) (default - [do not set](#)).

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

When an [<input>](#) element's [type](#) attribute is in the [Reset Button](#) state, the rules in this section apply.

The [<input>](#) element [represents](#) a button that, when activated, resets the form. If the element has a [value](#) attribute, the button's label must be the value of that attribute; otherwise, it must be an implementation-defined string that means "Reset" or some such. The element is a [button](#). 

NOTE:

Since the default label is implementation-defined, and the width of the button typically depends on the button's label, the button's width can leak a few bits of fingerprintable information. These bits are likely to be strongly correlated to the identity of the user agent and the user's locale.

If the element is *mutable*, then the element's [activation behavior](#), if the element has a [form owner](#) and the element's [node document](#) is [fully active](#), is to [reset](#) the [form owner](#); otherwise, it is to do nothing.

If the element is not *mutable*, it has no [activation behavior](#).

Constraint validation: The element is [barred from constraint validation](#).

Bookkeeping details

- The [value](#) IDL attribute [applies](#) to this element and is in mode [default](#).
- The following content attributes must not be specified and [do not apply](#) to the element: [accept](#), [alt](#), [autocomplete](#), [checked](#), [dirname](#), [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), [formtarget](#), [height](#), [inputmode](#), [list](#), [max](#), [maxlength](#), [min](#), [minlength](#), [multiple](#), [pattern](#), [placeholder](#), [readonly](#), [required](#), [size](#), [src](#), [step](#), and [width](#).
- The following IDL attributes and methods [do not apply](#) to the element: [checked](#), [files](#), [list](#), [selectionStart](#), [selectionEnd](#), [selectionDirection](#), [valueAsDate](#), and [valueAsNumber](#) IDL attributes; [select\(\)](#), [setRangeText\(\)](#), [setSelectionRange\(\)](#), [stepDown\(\)](#), and [stepUp\(\)](#) methods.
- The input and change events [do not apply](#).

§ 4.10.5.1.21. BUTTON STATE (type=button)

Allowed [ARIA role attribute](#) values:

[‘button’](#) (default - [do not set](#)), [‘link’](#), [‘menuitem’](#), [‘menuitemcheckbox’](#), [‘menuitemradio’](#) or [‘radio’](#).

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

When an [<input>](#) element’s [type](#) attribute is in the [Button](#) state, the rules in this section apply.

The [<input>](#) element [represents](#) a button with no default behavior. A label for the button must be provided in the [value](#) attribute, though it may be the empty string. If the element has a [value](#) attribute, the button’s label must be the value of that attribute; otherwise, it must be the empty string. The element is a [button](#).

If the element is *mutable*, the element’s [activation behavior](#) is to do nothing.

If the element is not *mutable*, it has no [activation behavior](#).

Constraint validation: The element is [barred from constraint validation](#).

Bookkeeping details

- The [value](#) IDL attribute [applies](#) to this element and is in mode [default](#).
- The following content attributes must not be specified and [do not apply](#) to the element: [accept](#), [alt](#), [autocomplete](#), [checked](#), [dirname](#), [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), [formtarget](#), [height](#), [inputmode](#), [list](#), [max](#), [maxlength](#), [min](#), [minlength](#), [multiple](#), [pattern](#), [placeholder](#), [readonly](#), [required](#), [size](#), [src](#), [step](#), and [width](#).
- The following IDL attributes and methods [do not apply](#) to the element: [checked](#), [files](#), [list](#), [selectionStart](#), [selectionEnd](#), [selectionDirection](#), [valueAsDate](#), and [valueAsNumber](#) IDL attributes; [select\(\)](#), [setRangeText\(\)](#), [setSelectionRange\(\)](#), [stepDown\(\)](#), and [stepUp\(\)](#) methods.
- The input and change events [do not apply](#).

4.10.5.2. *Implementation notes regarding localization of form controls*

This section is non-normative.

The formats shown to the user in date, time, and number controls is independent of the format used for [form submission](#).

Browsers are encouraged to use user interfaces that present dates, times, and numbers according to the conventions of either the locale implied by the [<input>](#) element’s [language](#) or the user’s preferred locale. Using the page’s locale will ensure consistency with page-provided data.

EXAMPLE 472

For example, it would be confusing to users if an American English page claimed that a Cirque De Soleil show was going to be showing on 02/03, but their browser, configured to use the British English locale, only showed the date 03/02 in the ticket purchase date picker. Using the page’s locale would at least ensure that the date was presented in the same format everywhere. (There’s still a risk that the user would end up arriving a month late, of course, but there’s only so much that can be done about such cultural differences...)

4.10.5.3. *Common [<input>](#) element attributes*

These attributes only [apply](#) to an [<input>](#) element if its [type](#) attribute is in a state whose definition declares that the attribute [applies](#). When an attribute [doesn’t apply](#) to an [<input>](#) element, user agents must [ignore](#) the attribute, regardless of the requirements and definitions below.

4.10.5.3.1. THE [maxlength](#) AND [minlength](#) ATTRIBUTES

The [maxlength](#) attribute, when it [applies](#), is a [form control \[maxlength attribute\]\(#\)](#) controlled by the [<input>](#) element’s [dirty value flag](#).

The **minlength** attribute, when it [applies](#), is a [form control minlength attribute](#) controlled by the `<input>` element's [dirty value flag](#).

If the `<input>` element has a [maximum allowed value length](#), then the [code-unit length](#) of the value of the element's `value` attribute must be equal to or less than the element's [maximum allowed value length](#).

EXAMPLE 473

The following extract shows how a messaging client's text entry could be arbitrarily restricted to a fixed number of characters, thus forcing any conversation through this medium to be terse and discouraging intelligent discourse.

```
<label>What are you doing? <input name=status maxlength=140></label>
```

EXAMPLE 474

Here, a password is given a minimum length:

```
<p><label>Username: <input name=u required></label>
<p><label>Password: <input name=p required minlength=12></label>
```

§ 4.10.5.3.2. THE `size` ATTRIBUTE

The **size** attribute gives the number of characters that, in a visual rendering, the user agent is to allow the user to see while editing the element's [value](#).

The `size` attribute, if specified, must have a value that is a [valid non-negative integer](#) greater than zero.

If the attribute is present, then its value must be parsed using the [rules for parsing non-negative integers](#), and if the result is a number greater than zero, then the user agent should ensure that at least that many characters are visible.

The `size` IDL attribute is [limited to only non-negative numbers greater than zero](#) and has a default value of 20.

§ 4.10.5.3.3. THE `readonly` ATTRIBUTE

The **readonly** attribute is a [boolean attribute](#) that controls whether or not the user can edit the form control. When specified, the element is not *mutable*.

Constraint validation: If the `readonly` attribute is specified on an `<input>` element, the element is [barred from constraint validation](#).

NOTE:

The difference between `disabled` and `readonly` is that read-only controls are still focusable, so the user can still select the text and interact with it, whereas disabled controls are entirely non-interactive. (For this reason, only text controls can be made read-only: it wouldn't make sense for checkboxes or buttons, for instances.)

EXAMPLE 475

In the following example, the existing product identifiers cannot be modified, but they are still displayed as part of the form, for consistency with the row representing a new product (where the identifier is not yet filled in).

```
<form action="products.cgi" method="post" enctype="multipart/form-data">
  <table>
    <tr> <th> Product ID <th> Product name <th> Price <th> Action
    <tr>
      <td> <input readonly="readonly" name="1.pid" value="H412">
      <td> <input required="required" name="1.pname" value="Floor lamp Ulke">
      <td> $<input required="required" type="number" min="0" step="0.01" name="1.pprice"
value="49.99">
      <td> <button formnovalidate="formnovalidate" name="action"
value="delete:1">Delete</button>
    <tr>
      <td> <input readonly="readonly" name="2.pid" value="FG28">
      <td> <input required="required" name="2.pname" value="Table lamp Ulke">
      <td> $<input required="required" type="number" min="0" step="0.01" name="2.pprice"
value="24.99">
      <td> <button formnovalidate="formnovalidate" name="action"
value="delete:2">Delete</button>
    <tr>
      <td> <input required="required" name="3.pid" value="" pattern="[A-Z0-9]+">
      <td> <input required="required" name="3.pname" value="">
      <td> $<input required="required" type="number" min="0" step="0.01" name="3.pprice"
value="">
      <td> <button formnovalidate="formnovalidate" name="action"
value="delete:3">Delete</button>
    </table>
    <p> <button formnovalidate="formnovalidate" name="action" value="add">Add</button> </p>
    <p> <button name="action" value="update">Save</button> </p>
  </form>
```

§ 4.10.5.3.4. THE required ATTRIBUTE

The **required** attribute is a [boolean attribute](#). When specified, the element is *required*.

Constraint validation: If the element is *required*, and its value IDL attribute [applies](#) and is in the mode [value](#), and the element is *mutable*, and the element's [value](#) is the empty string, then the element is [suffering from being missing](#).

EXAMPLE 476

The following form has two required fields, one for an e-mail address and one for a password. It also has a third field that is only considered valid if the user types the same password in the password field and this third field.

```
<h1>Create new account</h1>
<form action="/newaccount" method=post
  oninput="up2.setCustomValidity(up2.value != up.value ? 'Passwords do not match.' : '')">
  <p>
    <label for="username">E-mail address:</label>
    <input id="username" type=email required name=un>
  <p>
    <label for="password1">Password:</label>
    <input id="password1" type=password required name=up>
  <p>
    <label for="password2">Confirm password:</label>
    <input id="password2" type=password name=up2>
  <p>
    <input type=submit value="Create account">
  </form>
```

EXAMPLE 477

For radio buttons, the required attribute is satisfied if any of the radio buttons in the [group](#) is selected. Thus, in the following example, any of the radio buttons can be checked, not just the one marked as required:

```
<fieldset>
  <legend>Did the movie pass the Bechdel test?</legend>
  <p><label><input type="radio" name="bechdel" value="no-characters"> No, there are not even
two female characters in the movie. </label>
  <p><label><input type="radio" name="bechdel" value="no-names"> No, the female characters
never talk to each other. </label>
  <p><label><input type="radio" name="bechdel" value="no-topic"> No, when female characters
talk to each other it's always about a male character. </label>
  <p><label><input type="radio" name="bechdel" value="yes" required> Yes. </label>
  <p><label><input type="radio" name="bechdel" value="unknown"> I don't know. </label>
</fieldset>
```

To avoid confusion as to whether a [radio button group](#) is required or not, authors are encouraged to specify the attribute on all the radio buttons in a group. Indeed, in general, authors are encouraged to avoid having radio button groups that do not have any initially checked controls in the first place, as this is a state that the user cannot return to, and is therefore generally considered a poor user interface.

§ 4.10.5.3.5. THE `multiple` ATTRIBUTE

The **multiple** attribute is a [boolean attribute](#) that indicates whether the user is to be allowed to specify more than one value.

EXAMPLE 478

The following extract shows how an e-mail client's "Cc" field could accept multiple e-mail addresses.

```
<label>Cc: <input type=email multiple name=cc></label>
```

If the user had, amongst many friends in their user contacts database, two friends "Arthur Dent" (with address "art@example.net") and "Adam Josh" (with address "adamjosh@example.net"), then, after the user has typed "a", the user agent might suggest these two e-mail addresses to the user.

The page could also link in the user's contacts database from the site:

```
<label>Cc: <input type=email multiple name=cc list=contacts></label>
...
<datalist id="contacts">
  <option value="hedral@damowmow.com">
  <option value="pillar@example.com">
  <option value="astrophy@cute.example">
  <option value="astronomy@science.example.org">
</datalist>
```

Suppose the user had entered "bob@example.net" into this text field, and then started typing a second e-mail address starting with "a". The user agent might show both the two friends mentioned earlier, as well as the "astrophy" and "astronomy" values given in the `<datalist>` element.

EXAMPLE 479

The following extract shows how an e-mail client's "Attachments" field could accept multiple files for upload.

```
<label>Attachments: <input type=file multiple name=att></label>
```

§ 4.10.5.3.6. THE `pattern` ATTRIBUTE

The **pattern** attribute specifies a regular expression against which the control's value, or, when the **multiple** attribute applies and is set, the control's values, are to be checked.

If specified, the attribute's value must match the JavaScript *Pattern* production. [ECMA-262]

If an `<input>` element has a **pattern** attribute specified, and the attribute's value, when compiled as a JavaScript regular expression with only the "u" flag specified, compiles successfully, then the resulting regular expression is the element's **compiled pattern regular expression**. If the element has no such attribute, or if the value doesn't compile successfully, then the element has no compiled pattern regular expression. [ECMA-262]

NOTE:

If the value doesn't compile successfully, user agents are encouraged to log this fact in a developer console, to aid debugging.

Constraint validation: If the element's [value](#) is not the empty string, and either the element's `multiple` attribute is not specified or it [does not apply](#) to the `<input>` element given its `type` attribute's current state, and the element has a [compiled pattern regular expression](#) but that regular expression does not match the entirety of the element's [value](#), then the element is [suffering from a pattern mismatch](#).

Constraint validation: If the element's [value](#) is not the empty string, and the element's `multiple` attribute is specified and [applies](#) to the `<input>` element, and the element has a [compiled pattern regular expression](#) but that regular expression does not match the entirety of each of the element's [values](#), then the element is [suffering from a pattern mismatch](#).

The [compiled pattern regular expression](#), when matched against a string, must have its start anchored to the start of the string and its end anchored to the end of the string.

NOTE:

This implies that the regular expression language used for this attribute is the same as that used in JavaScript, except that the `pattern` attribute is matched against the entire value, not just any subset (somewhat as if it implied a `^(?:` at the start of the pattern and a `)$` at the end).

When an `<input>` element has a `pattern` attribute specified, authors should provide a description of the pattern in text near the control. Authors may also include a `title` attribute to give a description of the pattern. User agents may use the contents of this attribute, if it is present, when informing the user that the pattern is not matched, or at any other suitable time, such as in a tooltip or read out by assistive technology when the control gains focus.

⚠Warning! Relying on the `title` attribute for the visual display of text content is currently discouraged as many user agents do not expose the attribute in an accessible manner as required by this specification (e.g., requiring a pointing device such as a mouse to cause a tooltip to appear, which excludes keyboard-only users and touch-only users, such as anyone with a modern phone or tablet).

EXAMPLE 480

For example, the following snippet:

```
<label> Part number:
  <input pattern="[0-9][A-Z]{3}" name="part"
    title="A part number is a digit followed by three uppercase letters."/>
</label>
```

...could cause the user agent to display an alert such as:

A part number is a digit followed by three uppercase letters.You cannot submit this form when the field is incorrect.

When a control has a `pattern` attribute, the `title` attribute, if used, must describe the pattern. Additional information could also be included, so long as it assists the user in filling in the control. Otherwise, assistive technology would be impaired.

EXAMPLE 481

For instance, if the `title` attribute contained the caption of the control, assistive technology could end up saying something like The text you have entered does not match the required pattern. Birthday, which is not useful.

user agents may still show the `title` in non-error situations (for example, as a tooltip when hovering over the control), so authors should be careful not to word `titles` as if an error has necessarily occurred.

§ 4.10.5.3.7. THE `min` AND `max` ATTRIBUTES

Some form controls can have explicit constraints applied limiting the allowed range of values that the user can provide.

Normally, such a range would be linear and continuous. A form control can **have a periodic domain**, however, in which case the form control's broadest possible range is finite, and authors can specify explicit ranges within it that span the boundaries.

EXAMPLE 482

Specifically, the broadest range of a `type=time` control is midnight to midnight (24 hours), and authors can set both continuous linear ranges (such as 9pm to 11pm) and discontinuous ranges spanning midnight (such as 11pm to 1am).

The `min` and `max` attributes indicate the allowed range of values for the element.

Their syntax is defined by the section that defines the `type` attribute's current state.

If the element has a `min` attribute, and the result of applying the [algorithm to convert a string to a number](#) to the value of the `min` attribute is a number, then that number is the element's **minimum**; otherwise, if the `type` attribute's current state defines a **default minimum**, then that is the [minimum](#); otherwise, the element has no [minimum](#).

The `min` attribute also defines the [step base](#).

If the element has a `max` attribute, and the result of applying the [algorithm to convert a string to a number](#) to the value of the `max` attribute is a number, then that number is the element's **maximum**; otherwise, if the `type` attribute's current state defines a **default maximum**, then that is the [maximum](#); otherwise, the element has no [maximum](#).

If the element does not [have a periodic domain](#), the `max` attribute's value (the [maximum](#)) must not be less than the `min` attribute's value (its [minimum](#)).

NOTE:

If an element that does not [have a periodic domain](#) has a [maximum](#) that is less than its [minimum](#), then so long as the element has a [value](#), it will either be [suffering from an underflow](#) or [suffering from an overflow](#).

An element **has a reversed range** if it [has a periodic domain](#) and its [maximum](#) is less than its [minimum](#).

An element **has range limitations** if it has a defined [minimum](#) or a defined [maximum](#).

How these range limitations apply depends on whether the element has a `multiple` attribute.

↪ If the element does not have a `multiple` attribute specified or if the `multiple` attribute **does not apply**

Constraint validation: When the element has a [minimum](#) and does not [have a reversed range](#), and the result of applying the [algorithm to convert a string to a number](#) to the string given by the element's [value](#) is a number, and the number obtained from that algorithm is less than the [minimum](#), the element is [suffering from an underflow](#).

Constraint validation: When the element has a [maximum](#) and does not [have a reversed range](#), and the result of applying the [algorithm to convert a string to a number](#) to the string given by the element's [value](#) is a number, and the number obtained from that algorithm is more than the [maximum](#), the element is [suffering from an overflow](#).

Constraint validation: When an element [has a reversed range](#), and the result of applying the [algorithm to convert a string to a number](#) to the string given by the element's [value](#) is a number, and the number obtained from that algorithm is more than the [maximum](#) and less than the [minimum](#), the element is simultaneously [suffering from an underflow](#) and [suffering from an overflow](#).

↪ If the element does have a `multiple` attribute specified and the `multiple` attribute **does apply**

Constraint validation: When the element has a [minimum](#), and the result of applying the [algorithm to convert a string to a number](#) to any of the strings in the element's [values](#) is a number that is less than the [minimum](#), the element is [suffering from an underflow](#).

Constraint validation: When the element has a [maximum](#), and the result of applying the [algorithm to convert a string to a number](#) to any of the strings in the element's [values](#) is a number that is more than the [maximum](#), the element is [suffering from an overflow](#).

EXAMPLE 483

The following date control limits input to dates that are before the 1980s:

```
<input name=bday type=date max="1979-12-31">
```

EXAMPLE 484

The following number control limits input to whole numbers greater than zero:

```
<input name=quantity required="" type="number" min="1" value="1">
```

EXAMPLE 485

The following time control limits input to those minutes that occur between 9pm and 6am, defaulting to midnight:

```
<input name="sleepStart" type=time min="21:00" max="06:00" step="60" value="00:00">
```

§ 4.10.5.3.8. THE **step** ATTRIBUTE

The **step** attribute indicates the granularity that is expected (and required) of the [value](#) or [values](#), by limiting the allowed values. The section that defines the [type](#) attribute's current state also defines the **default step**, the **step scale factor**, and in some cases the **default step base**, which are used in processing the attribute as described below.

The [step](#) attribute, if specified, must either have a value that is a [valid floating-point number](#) that [parses](#) to a number that is greater than zero, or must have a value that is an [ASCII case-insensitive](#) match for the string "any".

The attribute provides the **allowed value step** for the element, as follows:

1. If the [step](#) attribute is absent, then the **allowed value step** is the [default step](#) multiplied by the [step scale factor](#).
2. Otherwise, if the attribute's value is an [ASCII case-insensitive](#) match for the string "any", then there is no **allowed value step**.
3. Otherwise, let *step value* be the result of running the [rules for parsing floating-point number values](#), when they are applied to the [step](#) attribute's value.
4. If the previous step returned an error, or *step value* is zero, or a number less than zero, then the **allowed value step** is the [default step](#) multiplied by the [step scale factor](#).
5. If the element's [type](#) attribute is in the [Date](#), [Month](#), [Week](#), or [Time](#) state, then round *step value* to the nearest whole number using the "round to nearest + round half up" technique, unless the value is less-than one, in which case let *step value* be 1.
6. The **allowed value step** is *step value* multiplied by the [step scale factor](#).

The **step base** is the value returned by the following algorithm:

1. If the element has a [min](#) content attribute, and the result of applying the [algorithm to convert a string to a number](#) to the value of the [min](#) content attribute is not an error, then return that result and abort these steps.
2. If the element has a [value](#) content attribute, and the result of applying the [algorithm to convert a string to a number](#) to the value of the [value](#) content attribute is not an error, then return that result and abort these steps.
3. If a [default step base](#) is defined for this element given its [type](#) attribute's state, then return it and abort these steps.
4. Return zero.

How these range limitations apply depends on whether the element has a [multiple](#) attribute.

➡ If the element does not have a [multiple](#) attribute specified or if the [multiple](#) attribute **does not apply**

Constraint validation: When the element has an [allowed value step](#), and the result of applying the [algorithm to](#)

[convert a string to a number](#) to the string given by the [value](#) is a number, and that number [is not step aligned](#), the element is [suffering from a step mismatch](#).

↪ **If the element does have a multiple attribute specified and the multiple attribute does [apply](#)**

Constraint validation: When the element has an [allowed value step](#), and the result of applying the [algorithm to convert a string to a number](#) to any of the strings in the [values](#) is a number that [is not step aligned](#), the element is [suffering from a step mismatch](#).

EXAMPLE 486

The following range control only accepts values in the range 0..1, and allows 256 steps in that range:

```
<input name=opacity type=range min=0 max=1 step=0.00392156863>
```

EXAMPLE 487

The following control allows any time in the day to be selected, with any accuracy (e.g., thousandth-of-a-second accuracy or more):

```
<input name=favtime type=time step=any>
```

Normally, time controls are limited to an accuracy of one minute.

§ 4.10.5.3.9. THE `list` ATTRIBUTE

The **`list`** attribute is used to identify an element that lists predefined options suggested to the user.

If present, its value must be the [ID](#) of a `<datalist>` element in the same document.

The **suggestions source element** is the first element in the document in [tree order](#) to have an [ID](#) equal to the value of the `list` attribute, if that element is a `<datalist>` element. If there is no `list` attribute, or if there is no element with that [ID](#), or if the first element with that [ID](#) is not a `<datalist>` element, then there is no [suggestions source element](#).

If there is a [suggestions source element](#), then, when the user agent is allowing the user to edit the `<input>` element's [value](#), the user agent should offer the suggestions represented by the [suggestions source element](#) to the user in a manner suitable for the type of control used. The user agent may use the suggestion's [label](#) to identify the suggestion if appropriate.

User agents are encouraged to filter the suggestions represented by the [suggestions source element](#) when the number of suggestions is large, including only the most relevant ones (e.g., based on the user's input so far). No precise threshold is defined, but capping the list at four to seven values is reasonable.

How user selections of suggestions are handled depends on whether the element is a control accepting a single value only, or whether it accepts multiple values:

↪ **If the element does not have a multiple attribute specified or if the multiple attribute [does not apply](#)**

When the user selects a suggestion, the `<input>` element's [value](#) must be set to the selected suggestion's [value](#), as if the user had written that value themselves.

↪ **If the element's `type` attribute is in the [Range](#) state and the element has a multiple attribute specified**

When the user selects a suggestion, the user agent must identify which value in the element's [values](#) the user intended to update, and must then update the element's [values](#) so that the relevant value is changed to the value given by the selected suggestion's [value](#), as if the user had themselves set it to that value.

↪ **If the element's `type` attribute is in the [E-mail](#) state and the element has a multiple attribute specified**

When the user selects a suggestion, the user agent must either add a new entry to the `<input>` element's [values](#), whose value is the selected suggestion's [value](#), or change an existing entry in the `<input>` element's [values](#) to have the value given by the selected suggestion's [value](#), as if the user had themselves added an entry with that value, or edited an existing entry to be that value. Which behavior is to be applied depends on the user interface in a user-agent-defined manner.



If the `list` attribute [does not apply](#), there is no [suggestions source element](#).

EXAMPLE 488

This URL field offers some suggestions.

```
<label>Homepage: <input name=hp type=url list=hpurls></label>
<datalist id=hpurls>
  <option value="https://www.google.com/" label="Google">
  <option value="https://www.reddit.com/" label="Reddit">
</datalist>
```

Other URLs from the user's history might show also; this is up to the user agent.

EXAMPLE 489

This example demonstrates how to design a form that uses the autocompletion list feature while still degrading usefully in legacy user agents.

If the autocompletion list is merely an aid, and is not important to the content, then simply using a `<datalist>` element with children `<option>` elements is enough. To prevent the values from being rendered in legacy user agents, they need to be placed inside the `value` attribute instead of inline.

```
<p>
  <label>
    Enter a breed:
    <input type="text" name="breed" list="breeds">
    <datalist id="breeds">
      <option value="Abyssinian">
      <option value="Alpaca">
      <!-- ... -->
    </datalist>
  </label>
</p>
```

However, if the values need to be shown in legacy user agents, then fallback content can be placed inside the `<datalist>` element, as follows:

```
<p>
  <label>
    Enter a breed:
    <input type="text" name="breed" list="breeds">
  </label>
  <datalist id="breeds">
    <label>
      or select one from the list:
      <select name="breed">
        <option value=""> (none selected)
        <option>Abyssinian
        <option>Alpaca
        <!-- ... -->
      </select>
    </label>
  </datalist>
</p>
```

The fallback content will only be shown in user agents that don't support `<datalist>`. The options, on the other hand, will be detected by all user agents, even though they are not children of the `<datalist>` element.

Note that if an `<option>` element used in a `<datalist>` is selected, it will be selected by default by legacy user agents (because it affects the `<select>`), but it will not have any effect on the `<input>` element in user agents that support `<datalist>`.

§ 4.10.5.3.10. THE `placeholder` ATTRIBUTE

The **placeholder** attribute represents a *short* hint (a word or short phrase) intended to aid the user with data entry when the control has no value. A hint could be a sample value or a brief description of the expected format. The attribute, if specified, must have a value that contains no U+000A LINE FEED (LF) or U+000D CARRIAGE RETURN (CR) characters.

The placeholder attribute should not be used as a replacement for a `<label>`. For a longer hint or other advisory text, place the text next to the control.

⚠Warning! Use of the `placeholder` attribute as a replacement for a `<label>` can reduce the accessibility and us-

ability of the control for a range of users including older users and users with cognitive, mobility, fine motor skill or vision impairments. While the hint given by the control's `<label>` is shown at all times, the short hint given in the placeholder attribute is only shown before the user enters a value. Furthermore, placeholder text may be mistaken for a pre-filled value, and as commonly implemented the default color of the placeholder text provides insufficient contrast and the lack of a separate visible `<label>` reduces the size of the hit region available for setting focus on the control.

User agents should present this hint to the user, after having [stripped line breaks](#) from it, when the element's `value` is the empty string, especially if the control is not [focused](#).

If a user agent normally doesn't show this hint to the user when the control is [focused](#), then the user agent should nonetheless show the hint for the control if it was focused as a result of the [autofocus](#) attribute, since in that case the user will not have had an opportunity to examine the control before focusing it.

EXAMPLE 490

Here is an example of a mail configuration user interface that uses the placeholder attribute:

```
<fieldset>
  <legend>Mail Account</legend>
  <p><label>Name: <input type="text" name="fullname" placeholder="John Ratzenberger">
</label></p>
  <p><label>Address: <input type="email" name="address" placeholder="john@example.net">
</label></p>
  <p><label>Password: <input type="password" name="password"></label></p>
  <p><label>Description: <input type="text" name="desc" placeholder="My Email Account">
</label></p>
</fieldset>
```

EXAMPLE 491

In situations where the control's content has one directionality but the placeholder needs to have a different directionality, Unicode's bidirectional-algorithm formatting characters can be used in the attribute value:

```
<input name=t1 type=tel placeholder="&#x202B; 1 الهاتف رقم &#x202E;">
<input name=t2 type=tel placeholder="&#x202B; 2 الهاتف رقم &#x202E;">
```

For slightly more clarity, here's the same example using numeric character references instead of inline Arabic:

```
<input name=t1 type=tel
placeholder="&#x202B;&#1585;&#1602;&#1605; &#1575;&#1604;&#1607;&#1575;&#1578;&#1601; 1&#x202E;">
<input name=t2 type=tel
placeholder="&#x202B;&#1585;&#1602;&#1605; &#1575;&#1604;&#1607;&#1575;&#1578;&#1601; 2&#x202E;">
```

4.10.5.4. Common `<input>` element APIs

This definition is non-normative. Implementation requirements are given below this definition.

`input . value` [= `value`]

Returns the current [value](#) of the form control.

Can be set, to change the value.

Throws an `"InvalidStateError"` [DOMException](#) if it is set to any value other than the empty string when the control is a [file upload](#) control.

`input . checked` [= `value`]

Returns the current [checkedness](#) of the form control.

Can be set, to change the [checkedness](#).

`input.files`

Returns a `FileList` object listing the [selected files](#) of the form control.

Returns null if the control isn't a file control.

`input.valueAsDate` [= `value`]

Returns a `Date` object representing the form control's [value](#), if applicable; otherwise, returns null.

Can be set, to change the value.

Throws an `"InvalidStateError"` [DOMException](#) if the control isn't date- or time-based.

`input.valueAsNumber` [= `value`]

Returns a number representing the form control's [value](#), if applicable; otherwise, returns NaN.

Can be set, to change the value. Setting this to NaN will set the underlying value to the empty string.

Throws an `"InvalidStateError"` [DOMException](#) if the control is neither date- or time-based nor numeric.

`input.stepUp` ([`n`])

`input.stepDown` ([`n`])

Changes the form control's [value](#) by the value given in the [step](#) attribute, multiplied by `n`. The default value for `n` is 1.

Throws `"InvalidStateError"` [DOMException](#) if the control is neither date- or time-based nor numeric, or if the [step](#) attribute's value is "any".

`input.list`

Returns the `<datalist>` element indicated by the `list` attribute.

The **value** IDL attribute allows scripts to manipulate the [value](#) of an `<input>` element. The attribute is in one of the following modes, which define its behavior:

value

On getting, it must return the current [value](#) of the element. On setting, it must set the element's [value](#) to the new value, set the element's [dirty value flag](#) to true, invoke the [value sanitization algorithm](#), if the element's [type](#) attribute's current state defines one, and then, if the element has a text entry cursor position, should move the text entry cursor position to the end of the text field, unselecting any selected text and resetting the selection direction to *none*.

default

On getting, if the element has a [value](#) attribute, it must return that attribute's value; otherwise, it must return the empty string. On setting, it must set the element's [value](#) attribute to the new value.

default/on

On getting, if the element has a [value](#) attribute, it must return that attribute's value; otherwise, it must return the string "on". On setting, it must set the element's [value](#) attribute to the new value.

filename

On getting, it must return the string "C:\fakepath\" followed by the name of the first file in the list of [selected files](#), if any, or the empty string if the list is empty. On setting, if the new value is the empty string, it must empty the list of [selected files](#); otherwise, it must throw an `"InvalidStateError"` [DOMException](#).

NOTE:

This "fakepath" requirement is a sad accident of history. See the example in the [File Upload](#) state section for more information.

NOTE:

Since [path components](#) are not permitted in file names in the list of [selected files](#), the "\fakepath\" cannot be mistaken for a path component.



The **checked** IDL attribute allows scripts to manipulate the [checkedness](#) of an `<input>` element. On getting, it must return the current [checkedness](#) of the element; and on setting, it must set the element's [checkedness](#) to the new value and set the element's [dirty checkedness flag](#) to true.



The **files** IDL attribute allows scripts to access the element's [selected files](#). On getting, if the IDL attribute [applies](#), it must return a `FileList` object that represents the current [selected files](#). The same object must be returned until the list of [selected files](#) changes. If the IDL attribute [does not apply](#), then it must instead return null. [\[FILEAPI\]](#)



The **valueAsDate** IDL attribute represents the [value](#) of the element, interpreted as a date.

On getting, if the `valueAsDate` attribute [does not apply](#), as defined for the `<input>` element's [type](#) attribute's current state, then return null. Otherwise, run the [algorithm to convert a string to a Date object](#) defined for that state to the element's [value](#); if the algorithm returned a [Date](#) object, then return it, otherwise, return null.

On setting, if the `valueAsDate` attribute [does not apply](#), as defined for the `<input>` element's [type](#) attribute's current state, then throw an `InvalidStateError` exception; otherwise, if the new value is not null and not a [Date](#) object throw a `TypeError` exception; otherwise if the new value is null or a [Date](#) object representing the NaN time value, then set the [value](#) of the element to the empty string; otherwise, run the [algorithm to convert a Date object to a string](#), as defined for that state, on the new value, and set the [value](#) of the element to the resulting string.



The **valueAsNumber** IDL attribute represents the [value](#) of the element, interpreted as a number.

On getting, if the `valueAsNumber` attribute [does not apply](#), as defined for the `<input>` element's [type](#) attribute's current state, then return a Not-a-Number (NaN) value. Otherwise, if the `valueAsDate` attribute [applies](#), run the [algorithm to convert a string to a Date object](#) defined for that state to the element's [value](#); if the algorithm returned a [Date](#) object, then return the *time value* of the object (the number of milliseconds from midnight UTC the morning of 1970-01-01 to the time represented by the [Date](#) object), otherwise, return a Not-a-Number (NaN) value. Otherwise, run the [algorithm to convert a string to a number](#) defined for that state to the element's [value](#); if the algorithm returned a number, then return it, otherwise, return a Not-a-Number (NaN) value.

On setting, if the new value is infinite, then throw a `TypeError` exception. Otherwise, if the `valueAsNumber` attribute [does not apply](#), as defined for the `<input>` element's [type](#) attribute's current state, then throw an `InvalidStateError` exception. Otherwise, if the new value is a Not-a-Number (NaN) value, then set the [value](#) of the element to the empty string. Otherwise, if the `valueAsDate` attribute [applies](#), run the [algorithm to convert a Date object to a string](#) defined for that state, passing it a [Date](#) object whose *time value* is the new value, and set the [value](#) of the element to the resulting string. Otherwise, run the [algorithm to convert a number to a string](#), as defined for that state, on the new value, and set the [value](#) of the element to the resulting string.



The **stepDown(*n*)** and **stepUp(*n*)** methods, when invoked, must run the following algorithm:

1. If the `stepDown()` and `stepUp()` methods [do not apply](#), as defined for the `<input>` element's [type](#) attribute's current state, then throw an `"InvalidStateError" DOMException`, and abort these steps.
2. If the element has no [allowed value step](#), then throw an `"InvalidStateError" DOMException`, and abort these steps.
3. If the element has a [minimum](#) and a [maximum](#) and the [minimum](#) is greater than the [maximum](#), then abort these steps.

4. If the element has a [minimum](#) and a [maximum](#) and there is no [step-aligned](#) value greater than or equal to the element's [minimum](#) and less than or equal to the element's [maximum](#), then abort these steps.
5. If applying the [algorithm to convert a string to a number](#) to the string given by the element's [value](#) does not result in an error, then let [value](#) be the result of that algorithm. Otherwise, let [value](#) be zero.
6. Let [valueBeforeStepping](#) be [value](#).
7. If [value](#) is not [step-aligned](#), then:
 1. If the method invoked was the `stepDown()` method, then [step-align](#) [value](#) with *negative preference*. Otherwise [step-align](#) [value](#) with *positive preference*. In either case, let [value](#) be the result.

EXAMPLE 492

This ensures that the value first snaps to a [step-aligned](#) value when it doesn't start step-aligned. For example, starting with the following `<input>` with [value](#) of 3:

```
<input type="number" value="3" min="1" max="10" step="2.6">
```

Invoking the `stepUp()` method will snap the [value](#) to 3.6; subsequent invocations will increment the value by 2.6 (e.g., 6.2, then 8.8). Likewise, the following `<input>` element in the [Week](#) state will also [step-align](#) in similar fashion, though in this state, the [step](#) value is rounded to 3, per the derivation of the [allowed value step](#).

```
<input type="week" value="2016-W20" min="2016-W01" max="2017-W01" step="2.6">
```

Invoking `stepUp()` will result in a [value](#) of "2016-W22" because the nearest [step-aligned](#) value from the [step base](#) of "2016-W01" (the [min](#) value) with 3 week [steps](#) that is greater than the [value](#) of "2016-W20" is "2016-W22" (i.e.: W01, W04, W07, W10, W13, W16, W19, W22).

Otherwise ([value](#) is [step-aligned](#)), run the following substeps:

1. Let n be the argument.
2. Let Δ be the [allowed value step](#) multiplied by n .
3. If the method invoked was the `stepDown()` method, negate Δ .
4. Let [value](#) be the result of adding Δ to [value](#).
8. If the element has a [minimum](#), and [value](#) is less than that [minimum](#), then set [value](#) to the [step-aligned minimum](#) value with *positive preference*.
9. If the element has a [maximum](#), and [value](#) is greater than that [maximum](#), then set [value](#) to the [step-aligned maximum](#) value with *negative preference*.
10. If either the method invoked was the `stepDown()` method and [value](#) is greater than [valueBeforeStepping](#), or the method invoked was the `stepUp()` method and [value](#) is less than [valueBeforeStepping](#), then abort these steps.

EXAMPLE 493

This ensures that invoking the `stepUp()` method on the `<input>` element in the following example does not change the [value](#) of that element:

```
<input type="number" value=1 max=0>
```

11. Let [value as string](#) be the result of running the [algorithm to convert a number to a string](#), as defined for the `<input>` element's [type](#) attribute's current state, on [value](#).
12. Set the [value](#) of the element to [value as string](#).

To determine if a value `v` is **step aligned** do the following:

NOTE:

This algorithm checks to see if a value falls along an `<input>` element's defined [step](#) intervals, with the interval's origin at the [step base](#) value. It is used to determine if the element's [value](#) is [suffering from a step mismatch](#) and for various checks in the [stepUp\(\)](#) and [stepDown\(\)](#) methods.

1. Subtract the [step base](#) from `v` and let the result be `relative distance`.
2. If dividing the `relative distance` by the [allowed value step](#) results in a value with a remainder then `v` is **not step aligned**. Otherwise it is step aligned.

To **step-align** a value `v` with either *negative preference* or *positive preference*, do the following:

NOTE:

negative preference selects a [step-aligned](#) value that is less than or equal to `v`, while *positive preference* [step-aligns](#) with a value greater than or equal to `v`.

1. Subtract the [step base](#) from `v` and let the result be `relative distance`.
2. Let `step interval count` be the result of integer dividing (or divide and throw out any remainder) `relative distance` by the [allowed value step](#).
3. Let `candidate` be the `step interval count` multiplied by the [allowed value step](#).
4. If this algorithm was invoked with *negative preference* and the value of `v` is less than `candidate`, then decrement `candidate` by the [allowed value step](#).

Otherwise, if this algorithm was invoked with *positive preference* and the value of `v` is greater than `candidate`, then increment `candidate` by the [allowed value step](#).

5. The [step-aligned](#) value is `candidate`. Return `candidate`.



The **list** IDL attribute must return the current [suggestions source element](#), if any, or null otherwise.

4.10.5.5. Common event behaviors

When the input and change events [apply](#) (which is the case for all `<input>` controls other than [buttons](#) and those with the [type](#) attribute in the [Hidden](#) state), the events are fired to indicate that the user has interacted with the control. The **input** event fires whenever the user has modified the data of the control. The **change** event fires when the value is committed, if that makes sense for the control, or else when the control [loses focus](#). In all cases, the input event comes before the corresponding change event (if any).

When an `<input>` element has a defined [activation behavior](#), the rules for dispatching these events, if they [apply](#), are given in the section above that defines the [type](#) attribute's state. (This is the case for all `<input>` controls with the [type](#) attribute in the [Checkbox](#) state, the [Radio Button](#) state, or the [File Upload](#) state.)

For `<input>` elements without a defined [activation behavior](#), but to which these events [apply](#), and for which the user interface involves both interactive manipulation and an explicit commit action, then when the user changes the element's [value](#), the user agent must [queue a task to fire a simple event](#) that bubbles named input at the `<input>` element, and any time the user commits the change, the user agent must [queue a task to fire a simple event](#) that bubbles named change at the `<input>` element.

EXAMPLE 494

An example of a user interface involving both interactive manipulation and a commit action would be a [Range](#) controls that use a slider, when manipulated using a pointing device. While the user is dragging the control's knob, input events would fire whenever the position changed, whereas the change event would only fire when the user let go of the knob, committing to a specific value.

For `<input>` elements without a defined [activation behavior](#), but to which these events [apply](#), and for which the user interface involves an explicit commit action but no intermediate manipulation, then any time the user commits a change to the element's [value](#), the user agent must [queue a task](#) to first [fire a simple event](#) that bubbles named input at the `<input>` element, and then [fire a simple event](#) that bubbles named change at the `<input>` element.

EXAMPLE 495

An example of a user interface with a commit action would be a [Color](#) control that consists of a single button that brings up a color wheel: if the [value](#) only changes when the dialog is closed, then that would be the explicit commit action. On the other hand, if manipulating the control changes the color interactively, then there might be no commit action.

EXAMPLE 496

Another example of a user interface with a commit action would be a [Date](#) control that allows both text-based user input and user selection from a drop-down calendar: while text input does not have an explicit commit step, selecting a date from the drop down calendar and then dismissing the drop down would be a commit action.

EXAMPLE 497

The [Range](#) control is also an example of a user interface that has a commit action when used with a pointing device (rather than a keyboard): during the time that the pointing device starts manipulating the slider until the time that the slider is released, no commit action is taken (though input events are fired as the value is changed). Only after the slider is released is the commit action taken.

For `<input>` elements without a defined [activation behavior](#), but to which these events [apply](#), any time the user causes the element's [value](#) to change without an explicit commit action, the user agent must [queue a task](#) to [fire a simple event](#) that bubbles named input at the `<input>` element. The corresponding change event, if any, will be fired when the control [loses focus](#).

EXAMPLE 498

Examples of a user changing the element's [value](#) would include the user typing into a text field, pasting a new value into the field, or undoing an edit in that field. Some user interactions do not cause changes to the value, e.g., hitting the "delete" key in an empty text field, or replacing some text in the field with text from the clipboard that happens to be exactly the same text.

EXAMPLE 499

A [Range](#) control in the form of a slider that the user has [focused](#) and is interacting with using a keyboard would be another example of the user changing the element's [value](#) without a commit step.

In the case of [tasks](#) that just fire an input event, user agents may wait for a suitable break in the user's interaction before [queuing](#) the tasks; for example, a user agent could wait for the user to have not hit a key for 100ms, so as to only fire the event when the user pauses, instead of continuously for each keystroke.

When the user agent is to change an `<input>` element's [value](#) on behalf of the user (e.g., as part of a form prefilling feature), the user agent must [queue a task](#) to first update the [value](#) accordingly, then [fire a simple event](#) that bubbles named input at the `<input>` element, then [fire a simple event](#) that bubbles named change at the `<input>` element.

NOTE:

These events are not fired in response to changes made to the values of form controls by scripts. (This is to make it easier to update the values of form controls in response to the user manipulating the controls, without having to then filter out the script's own changes to avoid an infinite loop.)

The [task source](#) for these [tasks](#) is the [user interaction task source](#).

§ 4.10.6. The button element

Categories:

[Flow content](#).

[Phrasing content](#).

[Interactive content](#).

[listed](#), [labelable](#), [submittable](#), and [reassociateable form-associated element](#).

[Palpable content](#).

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content](#), but there must be no [interactive content](#) descendant.

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

[autofocus](#) - Automatically focus the form control when the page is loaded

[disabled](#) - Whether the form control is disabled

[form](#) - Associates the control with a [<form>](#) element

[formaction](#) - [URL](#) to use for [§4.10.22 Form submission](#)

[formenctype](#) - Form data set encoding type to use for [§4.10.22 Form submission](#)

[formmethod](#) - HTTP method to use for [§4.10.22 Form submission](#)

[formnovalidate](#) - Bypass form control validation for [§4.10.22 Form submission](#)

[formtarget](#) - [browsing context](#) for [§4.10.22 Form submission](#)

[menu](#) - Specifies the element's [designated pop-up menu](#)

[name](#) - Name of form control to use for [§4.10.22 Form submission](#) and in the [form.elements](#) API

[type](#) - Type of button

[value](#) - Value to be used for [§4.10.22 Form submission](#)

Allowed ARIA role attribute values:

[‘button’](#) (default - *do not set*), [‘link’](#), [‘menuitem’](#), [‘menuitemcheckbox’](#), [‘menuitemradio’](#) or [‘radio’](#).

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any [aria-*](#) attributes [applicable to the allowed roles](#).

DOM interface:

```

interface HTMLButtonElement : HTMLElement {
    attribute boolean autofocus;
    attribute boolean disabled;
    readonly attribute HTMLFormElement? form;
    attribute DOMString formAction;
    attribute DOMString formEnctype;
    attribute DOMString formMethod;
    attribute boolean formNoValidate;
    attribute DOMString formTarget;
    attribute DOMString name;
    attribute DOMString type;
    attribute DOMString value;
    attribute HTMLMenuElement? menu;

    readonly attribute boolean willValidate;
    readonly attribute ValidityState validity;
    readonly attribute DOMString validationMessage;
    boolean checkValidity();
    boolean reportValidity();
    void setCustomValidity(DOMString error);

    [SameObject] readonly attribute NodeList labels;
};

```

The `<button>` element [represents](#) a button labeled by its contents.

The element is a [button](#).

The **type** attribute controls the behavior of the button when it is activated. It is an [enumerated attribute](#). The following table lists the keywords and states for the attribute — the keywords in the left column map to the states in the cell in the second column on the same row as the keyword.

Keyword	State	Brief description
submit	submit button	Submits the form.
reset	reset button	Resets the form.
button	Button	Does nothing.
menu	Menu	Shows a menu.

The *missing value default* is the [submit button](#) state.

If the **type** attribute is in the [submit button](#) state, the element is specifically a [submit button](#).

Constraint validation: If the **type** attribute is in the [reset button](#) state, the [Button](#) state, or the [Menu](#) state, the element is barred from constrain validation.

When a `<button>` element is not disabled, its [activation behavior](#) element is to run the steps defined in the following list for the current state of the element's **type** attribute:

[submit button](#)

If the element has a [form owner](#) and the element's [node document](#) is [fully active](#), the element must [submit](#) the [form owner](#) from the `<button>` element.

[reset button](#)

If the element has a [form owner](#) and the element's [node document](#) is [fully active](#), the element must [reset](#) the [form owner](#).

[Button](#)

Do nothing.

[Menu](#)

The element must follow these steps:

1. If the `<button>` is not [being rendered](#), abort these steps.
2. If the `<button>` element's [node document](#) is not [fully active](#), abort these steps.
3. Let *menu* be the element's [designated pop-up menu](#), if any. If there isn't one, then abort these steps.
4. Fire a [trusted](#) event with the name `show` at *menu*, using the `RelatedEvent` interface, with the `relatedTarget` attribute initialized to the `<button>` element. The event must be cancelable.
5. If the event is not canceled, then [build and show](#) the menu for *menu*, with the `<button>` element as the subject.

The `form` attribute is used to explicitly associate the `<button>` element with its [form owner](#). The `name` attribute represents the element's name. The `disabled` attribute is used to make the control non-interactive and to prevent its value from being submitted. The `autofocus` attribute controls focus. The `formaction`, `formenctype`, `formmethod`, `formnovalidate`, and `formtarget` attributes are [attributes for form submission](#).

NOTE:

The `formnovalidate` attribute can be used to make submit buttons that do not trigger the constraint validation.

The `formaction`, `formenctype`, `formmethod`, `formnovalidate`, and `formtarget` must not be specified if the element's `type` attribute is not in the [submit button](#) state.

The `value` attribute gives the element's value for the purposes of [form submission](#). The element's [value](#) is the value of the element's `value` attribute, if there is one, or the empty string otherwise.

NOTE:

A button (and its value) is only included in the [form submission](#) if the button itself was used to initiate the [form submission](#).



If the element's `type` attribute is in the [Menu](#) state, the `menu` attribute must be specified to give the element's menu. The value must be the [ID](#) of a `<menu>` element in the same [home subtree](#) whose `type` attribute is in the [popup menu](#) state. The attribute must not be specified if the element's `type` attribute is not in the [Menu](#) state.

A `<button>` element's **designated pop-up menu** is the first element in the `<button>`'s [home subtree](#) whose ID is that given by the `<button>` element's `menu` attribute, if there is such an element and its `type` attribute is in the [popup menu](#) state; otherwise, the element has no [designated pop-up menu](#).



The `value` and `menu` IDL attributes must [reflect](#) the content attributes of the same name.

The `type` IDL attribute must [reflect](#) the content attribute of the same name, [limited to only known values](#).

The `willValidate`, `validity`, and `validationMessage` IDL attributes, and the `checkValidity()`, `reportValidity()`, and `setCustomValidity()` methods, are part of the [constraint validation API](#). The `labels` IDL attribute provides a list of the element's `<label>`s. The `autofocus`, `disabled`, `form`, and `name` IDL attributes are part of the element's forms API.

EXAMPLE 500

The following button is labeled "Show hint" and pops up a dialog box when activated:

```
<button type=button
  onclick="alert('This 15-20 minute piece was composed by George Gershwin.')">
  Show hint
</button>
```

§ 4.10.7. The `select` element

Categories:

[Flow content](#).

[Phrasing content](#).

[Interactive content](#).

[listed](#), [labelable](#), [submittable](#), [resettable](#), and [reassociateable form-associated element](#).

[Palpable content](#).

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

Zero or more [<option>](#), [<optgroup>](#), and [script-supporting elements](#).

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

[autofocus](#) - Automatically focus the form control when the page is loaded

[disabled](#) - Whether the form control is disabled

[form](#) - Associates the control with a [<form>](#) element

[multiple](#) - Whether to allow multiple values

[name](#) - Name of form control to use for [§4.10.22 Form submission](#) and in the [form.elements](#) API

[required](#) - Whether the control is required for [§4.10.22 Form submission](#)

[size](#) - Size of the control

Allowed ARIA role attribute values:

[‘listbox’](#) (default - *do not set*) or [‘menu’](#).

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any [aria-* attributes](#) [applicable to the allowed roles](#).

DOM interface:

```

interface HTMLSelectElement : HTMLElement {
    attribute DOMString autocomplete;
    attribute boolean autofocus;
    attribute boolean disabled;
    readonly attribute HTMLFormElement? form;
    attribute boolean multiple;
    attribute DOMString name;
    attribute boolean required;
    attribute unsigned long size;

    readonly attribute DOMString type;

    [SameObject] readonly attribute HTMLOptionsCollection options;
    attribute unsigned long length;
    getter Element? item(unsigned long index);
    HTMLOptionElement? namedItem(DOMString name);
    void add((HTMLOptionElement or HTMLOptGroupElement) element, optional (HTMLElement or
long)? before = null);
    void remove(); // ChildNode overload
    void remove(long index);
    setter void (unsigned long index, HTMLOptionElement? option);

    [SameObject] readonly attribute HTMLCollection selectedOptions;
    attribute long selectedIndex;
    attribute DOMString value;

    readonly attribute boolean willValidate;
    readonly attribute ValidityState validity;
    readonly attribute DOMString validationMessage;
    boolean checkValidity();
    boolean reportValidity();
    void setCustomValidity(DOMString error);

    [SameObject] readonly attribute NodeList labels;
};

```

The `<select>` element represents a control for selecting amongst a set of options.

The **multiple** attribute is a [boolean attribute](#). If the attribute is present, then the `<select>` element [represents](#) a control for selecting zero or more options from the [list of options](#). If the attribute is absent, then the `<select>` element [represents](#) a control for selecting a single option from the [list of options](#).

The **size** attribute gives the number of options to show to the user. The size attribute, if specified, must have a value that is a [valid non-negative integer](#) greater than zero.

The **display size** of a `<select>` element is the result of applying the [rules for parsing non-negative integers](#) to the value of element's size attribute, if it has one and parsing it is successful. If applying those rules to the attribute's value is not successful, or if the size attribute is absent, then the element's [display size](#) is 4 if the element's **multiple** content attribute is present, and 1 otherwise.

The **list of options** for a `<select>` element consists of all the `<option>` element children of the `<select>` element, and all the `<option>` element children of all the `<optgroup>` element children of the `<select>` element, in [tree order](#).

The **required** attribute is a [boolean attribute](#). When specified, the user will be required to select a value before submitting the form.

If a `<select>` element has a required attribute specified, does not have a multiple attribute specified, and has a [display size](#) of 1; and if the [value](#) of the first `<option>` element in the `<select>` element's [list of options](#) (if any) is the empty string, and that `<option>` element's parent node is the `<select>` element (and not an `<optgroup>` element), then that `<option>` is the `<select>` element's **placeholder label option**.

If a `<select>` element has a required attribute specified, does not have a multiple attribute specified, and has a [display](#)

size of 1, then the `<select>` element must have a [placeholder label option](#).

NOTE:

In practice, the requirement stated in the paragraph above can only apply when a `<select>` element does not have a `sizes` attribute with a value greater than 1.

Constraint validation: If the element has its `required` attribute specified, and either none of the `<option>` elements in the `<select>` element's [list of options](#) have their `selectedness` set to true, or the only `<option>` element in the `<select>` element's [list of options](#) with its `selectedness` set to true is the [placeholder label option](#), then the element is [suffering from being missing](#).

If the `multiple` attribute is absent, and the element is not disabled, then the user agent should allow the user to pick an `<option>` element in its [list of options](#) that is itself not disabled. Upon this `<option>` element being **picked** (either through a click, or through unfocusing the element after changing its value, or through a [menu command](#), or through any other mechanism), and before the relevant user interaction event is queued (e.g., before the `click` event), the user agent must set the `selectedness` of the picked `<option>` element to true, set its `dirty` to true, and then [send select update notifications](#).

If the `multiple` attribute is absent, whenever an `<option>` element in the `<select>` element's [list of options](#) has its `selectedness` set to true, and whenever an `<option>` element with its `selectedness` set to true is added to the `<select>` element's [list of options](#), the user agent must set the `selectedness` of all the other `<option>` elements in its [list of options](#) to false.

If the `multiple` attribute is absent and the element's `display size` is greater than 1, then the user agent should also allow the user to request that the `<option>` whose `selectedness` is true, if any, be unselected. Upon this request being conveyed to the user agent, and before the relevant user interaction event is queued (e.g., before the `click` event), the user agent must set the `selectedness` of that `<option>` element to false, set its `dirty` to true, and then [send select update notifications](#).

If [nodes are inserted](#) or [nodes are removed](#) causing the [list of options](#) to gain or lose one or more `<option>` elements, or if an `<option>` element in the [list of options](#) [asks for a reset](#), then, if the `<select>` element's `multiple` attribute is absent, the user agent must run the first applicable set of steps from the following list:

- ↪ If the `<select>` element's `display size` is 1, and no `<option>` elements in the `<select>` element's [list of options](#) have their `selectedness` set to true

Set the `selectedness` of the first `<option>` element in the [list of options](#) in [tree order](#) that is not disabled, if any, to true.

- ↪ If two or more `<option>` elements in the `<select>` element's [list of options](#) have their `selectedness` set to true

Set the `selectedness` of all but the last `<option>` element with its `selectedness` set to true in the [list of options](#) in [tree order](#) to false.

If the `multiple` attribute is present, and the element is not disabled, then the user agent should allow the user to **toggle** the `selectedness` of the `<option>` elements in its [list of options](#) that are themselves not disabled. Upon such an element being **toggled** (either through a click, or through a [menu command](#), or any other mechanism), and before the relevant user interaction event is queued (e.g., before a related `click` event), the `selectedness` of the `<option>` element must be changed (from true to false or false to true), the `dirty` of the element must be set to true, and the user agent must [send select update notifications](#).

When the user agent is to **send select update notifications**, [queue a task](#) to first [fire a simple event](#) that bubbles named `input` at the `<select>` element, and then [fire a simple event](#) that bubbles named `change` at the `<select>` element, using the [user interaction task source](#) as the task source. If the [JavaScript execution context stack](#) was not empty when the user agent was to [send select update notifications](#), then the resulting `input` and `change` events must not be [trusted](#).

The [reset algorithm](#) for `<select>` elements is to go through all the `<option>` elements in the element's [list of options](#), set their `selectedness` to true if the `<option>` element has a `selected` attribute, and false otherwise, set their `dirty` to false, and then have the `<option>` elements [ask for a reset](#).

The `form` attribute is used to explicitly associate the `<select>` element with its [form owner](#). The `name` attribute represents the element's name. The `disabled` attribute is used to make the control non-interactive and to prevent its value from being submitted. The `autofocus` attribute controls focus. The `autocomplete` attribute controls how the user agent provides autofill behavior.

A `<select>` element that is not disabled is *mutable*.

This definition is non-normative. Implementation requirements are given below this definition.

`select` . `type`

Returns "select-multiple" if the element has a `multiple` attribute, and "select-one" otherwise.

`select` . `options`

Returns an `HTMLOptionsCollection` of the [list of options](#).

`select` . `length` [= `value`]

Returns the number of elements in the [list of options](#).

When set to a smaller number, truncates the number of `<option>` elements in the `<select>`.

When set to a greater number, adds new blank `<option>` elements to the `<select>`.

`element` = `select` . `item`(`index`)

`select` [`index`]

Returns the item with index `index` from the [list of options](#). The items are sorted in [tree order](#).

`element` = `select` . `namedItem`(`name`)

Returns the first item with `ID` or `name` `name` from the [list of options](#).

Returns null if no element with that `ID` could be found.

`select` . `add`(`element` [, `before`])

Inserts `element` before the node given by `before`.

The `before` argument can be a number, in which case `element` is inserted before the item with that number, or an element from the [list of options](#), in which case `element` is inserted before that element.

If `before` is omitted, null, or a number out of range, then `element` will be added at the end of the list.

This method will throw a `HierarchyRequestError` exception if `element` is an ancestor of the element into which it is to be inserted.

`select` . `selectedOptions`

Returns an `HTMLCollection` of the [list of options](#) that are selected.

`select` . `selectedIndex` [= `value`]

Returns the index of the first selected item, if any, or -1 if there is no selected item.

Can be set, to change the selection.

`select` . `value` [= `value`]

Returns the [value](#) of the first selected item, if any, or the empty string if there is no selected item.

Can be set, to change the selection.

The **`type`** IDL attribute, on getting, must return the string "select-one" if the `multiple` attribute is absent, and the string "select-multiple" if the `multiple` attribute is present.

The **`options`** IDL attribute must return an `HTMLOptionsCollection` rooted at the `select` node, whose filter matches the elements in the [list of options](#).

The options collection is also mirrored on the `HTMLSelectElement` object. The [supported property indices](#) at any instant are the indices supported by the object returned by the `options` attribute at that instant.

The **`length`** IDL attribute must return the number of nodes [represented](#) by the options collection. On setting, it must act like the attribute of the same name on the options collection.

The **`item`(`index`)** method must return the value returned by [the method of the same name](#) on the options collection, when invoked with the same argument.

The **`namedItem`(`name`)** method must return the value returned by [the method of the same name](#) on the options collection, when invoked with the same argument.

When the user agent is to **set the value of a new indexed property** for a given property index `index` to a new value `value`,

it must instead [set the value of a new indexed property](#) with the given property index *index* to the new value *value* on the options collection.

Similarly, the **add()** method must act like its namesake method on that same options collection.

The **remove()** method must act like its namesake method on that same options collection when it has arguments, and like its namesake method on the `ChildNode` interface implemented by the [HTMLSelectElement](#) ancestor interface [Element](#) when it has no arguments.

The **selectedOptions** IDL attribute must return an [HTMLCollection](#) rooted at the [<select>](#) node, whose filter matches the elements in the [list of options](#) that have their [selectedness](#) set to true.

The **selectedIndex** IDL attribute, on getting, must return the [index](#) of the first [<option>](#) element in the [list of options](#) in [tree order](#) that has its [selectedness](#) set to true, if any. If there isn't one, then it must return -1.

On setting, the **selectedIndex** attribute must set the [selectedness](#) of all the [<option>](#) elements in the [list of options](#) to false, and then the [<option>](#) element in the [list of options](#) whose [index](#) is the given new value, if any, must have its [selectedness](#) set to true and its [dirtiness](#) set to true.

NOTE:

This can result in no element having a [selectedness](#) set to true even in the case of the [<select>](#) element having no [multiple](#) attribute and a [display size](#) of 1.

The **value** IDL attribute, on getting, must return the [value](#) of the first [<option>](#) element in the [list of options](#) in [tree order](#) that has its [selectedness](#) set to true, if any. If there isn't one, then it must return the empty string.

On setting, the **value** attribute must set the [selectedness](#) of all the [<option>](#) elements in the [list of options](#) to false, and then the first [<option>](#) element in the [list of options](#), in [tree order](#), whose [value](#) is equal to the given new value, if any, must have its [selectedness](#) set to true and its [dirtiness](#) set to true.

NOTE:

This can result in no element having a [selectedness](#) set to true even in the case of the [<select>](#) element having no [multiple](#) attribute and a [display size](#) of 1.

The **multiple**, **required**, and **size** IDL attributes must [reflect](#) the respective content attributes of the same name. The **size** IDL attribute has a default value of zero.

NOTE:

For historical reasons, the default value of the **size** IDL attribute does not return the actual size used, which, in the absence of the **size** content attribute, is either 1 or 4 depending on the presence of the **multiple** attribute.

The [willValidate](#), [validity](#), and [validationMessage](#) IDL attributes, and the [checkValidity\(\)](#), [reportValidity\(\)](#), and [setCustomValidity\(\)](#) methods, are part of the [constraint validation API](#). The [labels](#) IDL attribute provides a list of the element's [<label>](#)s. The [autofocus](#), [disabled](#), [form](#), and [name](#) IDL attributes are part of the element's forms API.

EXAMPLE 501

The following example shows how a `<select>` element can be used to offer the user with a set of options from which the user can select a single option. The default option is preselected.

```
<p>
  <label for="unitttype">Select unit type:</label>
  <select id="unitttype" name="unitttype">
    <option value="1"> Miner </option>
    <option value="2"> Puffer </option>
    <option value="3" selected> Snipey </option>
    <option value="4"> Max </option>
    <option value="5"> Firebot </option>
  </select>
</p>
```

When there is no default option, a value that provides instructions or a hint (placeholder option) can be used instead:

```
<select name="unitttype" required>
  <option value=""> Select unit type </option>
  <option value="1"> Miner </option>
  <option value="2"> Puffer </option>
  <option value="3"> Snipey </option>
  <option value="4"> Max </option>
  <option value="5"> Firebot </option>
</select>
```

EXAMPLE 502

Here, the user is offered a set of options from which he can select any number. By default, all five options are selected.

```
<p>
  <label for="allowedunits">Select unit types to enable on this map:</label>
  <select id="allowedunits" name="allowedunits" multiple>
    <option value="1" selected> Miner </option>
    <option value="2" selected> Puffer </option>
    <option value="3" selected> Snipey </option>
    <option value="4" selected> Max </option>
    <option value="5" selected> Firebot </option>
  </select>
</p>
```

EXAMPLE 503

Sometimes, a user has to select one or more items. This example shows such an interface.

```
<p>Select the songs from that you would like on your Act II Mix Tape:</p>
<select multiple required name="act2">
  <option value="s1">It Sucks to Be Me (Reprize)
  <option value="s2">There is Life Outside Your Apartment
  <option value="s3">The More You Ruv Someone
  <option value="s4">Schadenfreude
  <option value="s5">I Wish I Could Go Back to College
  <option value="s6">The Money Song
  <option value="s7">School for Monsters
  <option value="s8">The Money Song (Reprize)
  <option value="s9">There's a Fine, Fine Line (Reprize)
  <option value="s10">What Do You Do With a B.A. in English? (Reprize)
  <option value="s11">For Now
</select>
```

§ 4.10.8. The `datalist` element

Categories:

[Flow content.](#)

[Phrasing content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

Either: [phrasing content](#).

Or: Zero or more [<option>](#) and [script-supporting elements](#).

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[‘listbox’](#) (default - *do not set*).

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLDataListElement : HTMLElement {
  [SameObject] readonly attribute HTMLCollection options;
};
```

The `<datalist>` element represents a set of `<option>` elements that represent predefined options for other controls. In the rendering, the `<datalist>` element [represents](#) nothing and it, along with its children, should be hidden.

The `<datalist>` element can be used in two ways. In the simplest case, the `<datalist>` element has just `<option>` element children.

EXAMPLE 504

```
<label>
  Sex:
  <input name=sex list=sexes>
  <datalist id=sexes>
    <option value="Female">
    <option value="Male">
  </datalist>
</label>
```

In the more elaborate case, the `<datalist>` element can be given contents that are to be displayed for down-level clients that don't support `<datalist>`. In this case, the `<option>` elements are provided inside a `<select>` element inside the `<datalist>` element.

EXAMPLE 505

```

<label>
  Sex:
  <input name=sex list=sexes>
</label>
<datalist id=sexes>
  <label>
    or select from the list:
    <select name=sex>
      <option value="">
      <option>Female
      <option>Male
    </select>
  </label>
</datalist>

```

The [<datalist>](#) element is hooked up to an [<input>](#) element using the `list` attribute on the [<input>](#) element.

Each [<option>](#) element that is a descendant of the [<datalist>](#) element, that is not disabled, and whose [value](#) is a string that isn't the empty string, represents a suggestion. Each suggestion has a [value](#) and a [label](#).

This definition is non-normative. Implementation requirements are given below this definition.

***datalist*.options**

Returns an [HTMLCollection](#) of the [<option>](#) elements of the [<datalist>](#) element.

The **options** IDL attribute must return an [HTMLCollection](#) rooted at the [<datalist>](#) node, whose filter matches [<option>](#) elements.

Constraint validation: If an element has a [<datalist>](#) element ancestor, it is [barred from constraint validation](#).

§ 4.10.9. The optgroup element

Categories:

None.

Contexts in which this element can be used:

As a child of a [<select>](#) element.

Content model:

Zero or more [<option>](#) and [script-supporting elements](#).

Tag omission in text/html:

An [<optgroup>](#) element's [end tag](#) may be omitted if the [<optgroup>](#) element is immediately followed by another [<optgroup>](#) element, or if there is no more content in the parent element.

Content attributes:

[Global attributes](#)

[disabled](#) - Whether the form control is disabled

[label](#) - User-visible label

Allowed ARIA role attribute values:

None

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

DOM interface:

```
interface HTMLOptGroupElement : HTMLElement {
    attribute boolean disabled;
    attribute DOMString label;
};
```

The `<optgroup>` element [represents](#) a group of `<option>` elements with a common label.

The element's group of `<option>` elements consists of the `<option>` elements that are children of the `<optgroup>` element.

When showing `<option>` elements in `<select>` elements, user agents should show the `<option>` elements of such groups as being related to each other and separate from other `<option>` elements.

The **disabled** content attribute is a [boolean attribute](#) and can be used to [disable](#) a group of `<option>` elements together.

The **label** content attribute must be specified. Its value gives the name of the group, for the purposes of the user interface. User agents should use this attribute's value when labeling the group of `<option>` elements in a `<select>` element.

The **disabled** and **label** IDL attributes must [reflect](#) the respective content attributes of the same name.

NOTE:

There is no way to select an `<optgroup>` element. Only `<option>` elements can be selected. An `<optgroup>` element merely provides a label for a group of `<option>` elements.

EXAMPLE 506

The following snippet shows how a set of lessons from three courses could be offered in a `<select>` drop-down widget:

```
<form action="courseselector.dll" method="get">
  <p>Which course would you like to watch today?
  <p><label>Course:
  <select name="c">
    <optgroup label="8.01 Physics I: Classical Mechanics">
      <option value="8.01.1">Lecture 01: Powers of Ten
      <option value="8.01.2">Lecture 02: 1D Kinematics
      <option value="8.01.3">Lecture 03: Vectors
    <optgroup label="8.02 Electricity and Magnetism">
      <option value="8.02.1">Lecture 01: What holds our world together?
      <option value="8.02.2">Lecture 02: Electric Field
      <option value="8.02.3">Lecture 03: Electric Flux
    <optgroup label="8.03 Physics III: Vibrations and Waves">
      <option value="8.03.1">Lecture 01: Periodic Phenomenon
      <option value="8.03.2">Lecture 02: Beats
      <option value="8.03.3">Lecture 03: Forced Oscillations with Damping
  </select>
  </label>
  <p><input type="submit" value="► Play">
</form>
```

4.10.10. The option element

Categories:

None.

Contexts in which this element can be used:

- As a child of a `<select>` element.
- As a child of a `<datalist>` element.
- As a child of an `<optgroup>` element.

Content model:

If the element has a [label](#) attribute and a [value](#) attribute: [Nothing](#).

If the element has a [label](#) attribute but no [value](#) attribute: [Text](#).

If the element has no [label](#) attribute: [Text](#).

Tag omission in text/html:

An `<option>` element's [end tag](#) may be omitted if the `<option>` element is immediately followed by another `<option>` element, or if it is immediately followed by an `<optgroup>` element, or if there is no more content in the parent element.

Content attributes:

[Global attributes](#)

[disabled](#) - Whether the form control is disabled

[label](#) - User-visible label

[selected](#) - Whether the option is selected by default

[value](#) - Value to be used for [§4.10.22 Form submission](#)

Allowed ARIA role attribute values:

[‘option’](#) (default - *do not set*), [‘menuitem’](#), [‘menuitemradio’](#) or [‘separator’](#).

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
[NamedConstructor=Option(optional DOMString text = "", optional DOMString value,
optional boolean defaultSelected = false, optional boolean selected = false)]
interface HTMLOptionElement : HTMLElement {
    attribute boolean disabled;
    readonly attribute HTMLFormElement? form;
    attribute DOMString label;
    attribute boolean defaultSelected;
    attribute boolean selected;
    attribute DOMString value;

    attribute DOMString text;
    readonly attribute long index;
};
```

The `<option>` element [represents](#) an option in a `<select>` element or as part of a list of suggestions in a `<datalist>` element.

In certain circumstances described in the definition of the `<select>` element, an `<option>` element can be a `<select>` element's [placeholder label option](#). A [placeholder label option](#) does not represent an actual option, but instead represents a label for the `<select>` control.

The **disabled** content attribute is a [boolean attribute](#). An `<option>` element is disabled if its [disabled](#) attribute is present or if it is a child of an `<optgroup>` element whose [disabled](#) attribute is present.

An `<option>` element that is disabled must prevent any [click](#) events that are [queued](#) on the [user interaction task source](#) from being dispatched on the element.

The **label** content attribute provides a label for the element. The **label** of an `<option>` element is the value of the [label](#) content attribute, if there is one and its value is not the empty string, or, otherwise, the value of the element's [text](#) IDL attribute if its value is not the empty string.

The [label](#) content attribute, if specified, must not be empty.

The **value** content attribute provides a value for element. The *value* of an `<option>` element is the value of the **value** content attribute, if there is one, or, if there is not, the value of the element's [text](#) IDL attribute (which may be the empty string).

The **selected** content attribute is a [boolean attribute](#). It represents the default [selectedness](#) of the element.

The **dirty**ness of an `<option>` element is a boolean state, initially false. It controls whether adding or removing the [selected](#) content attribute has any effect.

The **selectedness** of an `<option>` element is a boolean state, initially false. Except where otherwise specified, when the element is created, its **selectedness** must be set to true if the element has a **selected** attribute. Whenever an `<option>` element's **selected** attribute is added, if its **dirtyiness** is false, its **selectedness** must be set to true. Whenever an `<option>` element's **selected** attribute is *removed*, if its **dirtyiness** is false, its **selectedness** must be set to false.

NOTE:

The `Option()` constructor, when called with three or fewer arguments, overrides the initial state of the **selectedness** state to always be false even if the third argument is true (implying that a **selected** attribute is to be set). The fourth argument can be used to explicitly set the initial **selectedness** state when using the constructor.

A `<select>` element whose **multiple** attribute is not specified must not have more than one descendant `<option>` element with its **selected** attribute set.

An `<option>` element's **index** is the number of `<option>` elements that are in the same **list of options** but that come before it in **tree order**. If the `<option>` element is not in a **list of options**, then the `<option>` element's **index** is zero.

This definition is non-normative. Implementation requirements are given below this definition.

`option` . `selected`

Returns true if the element is selected, and false otherwise.

Can be set, to override the current state of the element.

`option` . `index`

Returns the index of the element in its `<select>` element's **options** list.

`option` . `form`

Returns the element's `<form>` element, if any, or null otherwise.

`option` . `text`

Same as **textContent**, except that spaces are collapsed and `<script>` elements are skipped.

`option` = new `Option()`([`text` [, `value` [, `defaultSelected` [, `selected`]]]])

Returns a new `<option>` element.

The `text` argument sets the contents of the element.

The `value` argument sets the **value** attribute.

The `defaultSelected` argument sets the **selected** attribute.

The `selected` argument sets whether or not the element is selected. If it is omitted, even if the `defaultSelected` argument is true, the element is not selected.

The **disabled** IDL attribute must **reflect** the content attribute of the same name. The **defaultSelected** IDL attribute must **reflect** the **selected** content attribute.

The **label** IDL attribute, on getting, if there is a **label** content attribute, must return that attribute's value; otherwise, it must return the element's **label**. On setting, the element's **label** content attribute must be set to the new value.

The **value** IDL attribute, on getting, must return the element's **value**. On setting, the element's **value** content attribute must be set to the new value.

The **selected** IDL attribute, on getting, must return true if the element's **selectedness** is true, and false otherwise. On setting, it must set the element's **selectedness** to the new value, set its **dirtyiness** to true, and then cause the element to **ask for a reset**.

The **index** IDL attribute must return the element's **index**.

The **text** IDL attribute, on getting, must return the result of **stripping and collapsing whitespace** from the concatenation of **data** of all the **Text** node descendants of the `<option>` element, in **tree order**, excluding any that are descendants of descendants of the `<option>` element that are themselves `<script>` elements in the **HTML namespace** or `<script>` elements in the **SVG namespace**.

On setting, the **text** attribute must act as if the **textContent** IDL attribute on the element had been set to the new value.

The `form` IDL attribute's behavior depends on whether the `<option>` element is in a `<select>` element or not. If the `<option>` has a `<select>` element as its parent, or has an `<optgroup>` element as its parent and that `<optgroup>` element has a `<select>` element as its parent, then the `form` IDL attribute must return the same value as the `form` IDL attribute on that `<select>` element. Otherwise, it must return null.

A constructor is provided for creating `HTMLOptionElement` objects (in addition to the factory methods from DOM such as `createElement()`): `Option(text, value, defaultSelected, selected)`. When invoked as a constructor, it must return a new `HTMLOptionElement` object (a new `<option>` element). If the first argument is not the empty string, the new object must have as its only child a `Text` node whose data is the value of that argument. Otherwise, it must have no children. If the `value` argument is present, the new object must have a `value` attribute set with the value of the argument as its value. If the `defaultSelected` argument is true, the new object must have a `selected` attribute set with no value. If the `selected` argument is true, the new object must have its `selectedness` set to true; otherwise the `selectedness` must be set to false, even if the `defaultSelected` argument is true. The element's `node document` must be the `active document` of the `browsing context` of the `Window` object on which the interface object of the invoked constructor is found.

§ 4.10.11. The textarea element

Categories:

[Flow content](#),
[Phrasing content](#),
[Interactive content](#),
[listed, labelable, submittable, resettable, and reassociateable form-associated element](#),
[Palpable content](#).

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Text](#).

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)
[autocomplete](#) - Hint for form autofill feature
[autofocus](#) - Automatically focus the form control when the page is loaded
[cols](#) - Maximum number of characters per line
[dirname](#) - Name of form field to use for sending the element's [directionality](#) in [§4.10.22 Form submission](#)
[disabled](#) - Whether the form control is disabled
[form](#) - Associates the control with a `<form>` element
[inputmode](#) - Hint for selecting an input modality
[maxlength](#) - Maximum length of value
[minlength](#) - Minimum length of value
[name](#) - Name of form control to use for [§4.10.22 Form submission](#) and in the `form.elements` API
[placeholder](#) - User-visible label to be placed within the form control
[readonly](#) - Whether to allow the value to be edited by the user
[required](#) - Whether the control is required for [§4.10.22 Form submission](#)
[rows](#) - Number of lines to show
[wrap](#) - How the value of the form control is to be wrapped for [§4.10.22 Form submission](#)

Allowed ARIA role attribute values:

`'textbox'` (default - *do not set*).

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)
 Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```

interface HTMLTextAreaElement : HTMLElement {
    attribute DOMString autocomplete;
    attribute boolean autofocus;
    attribute unsigned long cols;
    attribute DOMString dirName;
    attribute boolean disabled;
    readonly attribute HTMLFormElement? form;
    attribute DOMString inputMode;
    attribute long maxLength;
    attribute long minLength;
    attribute DOMString name;
    attribute DOMString placeholder;
    attribute boolean readOnly;
    attribute boolean required;
    attribute unsigned long rows;
    attribute DOMString wrap;

    readonly attribute DOMString type;
    attribute DOMString defaultValue;
    [TreatNullAs=EmptyString] attribute DOMString value;
    readonly attribute unsigned long textLength;

    readonly attribute boolean willValidate;
    readonly attribute ValidityState validity;
    readonly attribute DOMString validationMessage;
    boolean checkValidity();
    boolean reportValidity();
    void setCustomValidity(DOMString error);

    [SameObject] readonly attribute NodeList labels;

    void select();
    attribute unsigned long selectionStart;
    attribute unsigned long selectionEnd;
    attribute DOMString selectionDirection;
    void setRangeText(DOMString replacement);
    void setRangeText(DOMString replacement, unsigned long start, unsigned long end,
optional SelectionMode selectionMode = "preserve");
    void setSelectionRange(unsigned long start, unsigned long end, optional DOMString
direction);
};

```

The `<textarea>` element [represents](#) a multiline plain text edit control for the element's **raw value**. The contents of the control represent the control's default value.

The [raw value](#) of a `<textarea>` control must be initially the empty string.

NOTE:

This element [has rendering requirements involving the bidirectional algorithm](#).

The **readonly** attribute is a [boolean attribute](#) used to control whether the text can be edited by the user or not.

EXAMPLE 507

In this example, a text field is marked read-only because it represents a read-only file:

```
Filename: <code>/etc/bash.bashrc</code>
<textarea name="buffer" readonly>
# System-wide .bashrc file for interactive bash(1) shells.

# To enable the settings / commands in this file for login shells as well,
# this file has to be sourced in /etc/profile.

# If not running interactively, don't do anything
[ -z "$PS1" ] &amp;& return

...</textarea>
```

Constraint validation: If the [readonly](#) attribute is specified on a [<textarea>](#) element, the element is [barred from constraint validation](#).

A [<textarea>](#) element is [mutable](#) if it is neither disabled nor has a [readonly](#) attribute specified.

When a [<textarea>](#) is [mutable](#), its [raw value](#) should be editable by the user: the user agent should allow the user to edit, insert, and remove text, and to insert and remove line breaks in the form of U+000A LINE FEED (LF) characters. Any time the user causes the element's [raw value](#) to change, the user agent must [queue a task to fire a simple event](#) that bubbles named input at the [<textarea>](#) element. User agents may wait for a suitable break in the user's interaction before queuing the task; for example, a user agent could wait for the user to have not hit a key for 100ms, so as to only fire the event when the user pauses, instead of continuously for each keystroke.

A [<textarea>](#) element has a **dirty value flag**, which must be initially set to false, and must be set to true whenever the user interacts with the control in a way that changes the [raw value](#).

When the [<textarea>](#) element's [textContent](#) IDL attribute changes value, if the element's [dirty value flag](#) is false, then the element's [raw value](#) must be set to the value of the element's [textContent](#) IDL attribute.

The [reset algorithm](#) for [<textarea>](#) elements is to set the element's [raw value](#) to the value of the element's [textContent](#) IDL attribute.

When a [<textarea>](#) element is popped off the [stack of open elements](#) of an [HTML parser](#) or [XML parser](#), then the user agent must invoke the element's [reset algorithm](#).

If the element is [mutable](#), the user agent should allow the user to change the writing direction of the element, setting it either to a left-to-right writing direction or a right-to-left writing direction. If the user does so, the user agent must then run the following steps:

1. Set the element's [dir](#) attribute to "[ltr](#)" if the user selected a left-to-right writing direction, and "[rtl](#)" if the user selected a right-to-left writing direction.
2. [Queue a task to fire a simple event](#) that bubbles named input at the [<textarea>](#) element.

The **cols** attribute specifies the expected maximum number of characters per line. If the **cols** attribute is specified, its value must be a [valid non-negative integer](#) greater than zero. If applying the [rules for parsing non-negative integers](#) to the attribute's value results in a number greater than zero, then the element's **character width** is that value; otherwise, it is 20.

The user agent may use the [<textarea>](#) element's [character width](#) as a hint to the user as to how many characters the server prefers per line (e.g., for visual user agents by making the width of the control be that many characters). In visual renderings, the user agent should wrap the user's input in the rendering so that each line is no wider than this number of characters.

The **rows** attribute specifies the number of lines to show. If the **rows** attribute is specified, its value must be a [valid non-negative integer](#) greater than zero. If applying the [rules for parsing non-negative integers](#) to the attribute's value results in a number greater than zero, then the element's **character height** is that value; otherwise, it is 2.

Visual user agents should set the height of the control to the number of lines given by [character height](#).

The **wrap** attribute is an [enumerated attribute](#) with two keywords and states: the **soft** keyword which maps to the [Soft](#) state, and the **hard** keyword which maps to the [Hard](#) state. The *missing value default* is the [Soft](#) state.

The **Soft** state indicates that the text in the `<textarea>` is not to be wrapped when it is submitted (though it can still be wrapped in the rendering).

The **Hard** state indicates that the text in the `<textarea>` is to have newlines added by the user agent so that the text is wrapped when it is submitted.

If the element's **wrap** attribute is in the [Hard](#) state, the **cols** attribute must be specified.

For historical reasons, the element's value is normalized in three different ways for three different purposes. The [raw value](#) is the value as it was originally set. It is not normalized. The [API value](#) is the value used in the **value** IDL attribute. It is normalized so that line breaks use U+000A LINE FEED (LF) characters. Finally, there is the [value](#), as used in [form submission](#) and other processing models in this specification. It is normalized so that line breaks use U+000D CARRIAGE RETURN U+000A LINE FEED (CRLF) character pairs, and in addition, if necessary given the element's **wrap** attribute, additional line breaks are inserted to wrap the text at the given width.

The element's **API value** is defined to be the element's [raw value](#) with the following transformation applied:

1. Replace every U+000D CARRIAGE RETURN U+000A LINE FEED (CRLF) character pair from the [raw value](#) with a single U+000A LINE FEED (LF) character.
2. Replace every remaining U+000D CARRIAGE RETURN character from the [raw value](#) with a single U+000A LINE FEED (LF) character.

The element's [value](#) is defined to be the element's [raw value](#) with the [textarea wrapping transformation](#) applied. The **textarea wrapping transformation** is the following algorithm, as applied to a string:

1. Replace every occurrence of a U+000D CARRIAGE RETURN (CR) character not followed by a U+000A LINE FEED (LF) character, and every occurrence of a U+000A LINE FEED (LF) character not preceded by a U+000D CARRIAGE RETURN (CR) character, by a two-character string consisting of a U+000D CARRIAGE RETURN U+000A LINE FEED (CRLF) character pair.
2. If the element's **wrap** attribute is in the [Hard](#) state, insert U+000D CARRIAGE RETURN U+000A LINE FEED (CRLF) character pairs into the string using a user agent-defined algorithm so that each line has no more than [character width](#) characters. For the purposes of this requirement, lines are delimited by the start of the string, the end of the string, and U+000D CARRIAGE RETURN U+000A LINE FEED (CRLF) character pairs.

The **maxlength** attribute is a [form control maxlength attribute](#) controlled by the `<textarea>` element's [dirty value flag](#).

If the `<textarea>` element has a [maximum allowed value length](#), then the element's children must be such that the [code-unit length](#) of the value of the element's **textContent** IDL attribute with the [textarea wrapping transformation](#) applied is equal to or less than the element's [maximum allowed value length](#).

The **minlength** attribute is a [form control minlength attribute](#) controlled by the `<textarea>` element's [dirty value flag](#).

The **required** attribute is a [boolean attribute](#). When specified, the user will be required to enter a value before submitting the form.

Constraint validation: If the element has its **required** attribute specified, and the element is [mutable](#), and the element's [value](#) is the empty string, then the element is [suffering from being missing](#).

The **placeholder** attribute represents a *short* hint (a word or short phrase) intended to aid the user with data entry when the control has no value. A hint could be a sample value or a brief description of the expected format.

The **placeholder** attribute should not be used as a replacement for a `<label>`. For a longer hint or other advisory text, place the text next to the control.

NOTE:

Use of the `placeholder` attribute as a replacement for a `<label>` can reduce the accessibility and usability of the control for a range of users including older users and users with cognitive, mobility, fine motor skill or vision impairments. While the hint given by the control's `<label>` is shown at all times, the short hint given in the `placeholder` attribute is only shown before the user enters a value. Furthermore, placeholder text may be mistaken for a pre-filled value, and as commonly implemented the default color of the placeholder text provides insufficient contrast and the lack of a separate visible `<label>` reduces the size of the hit region available for setting focus on the control.

User agents should present this hint to the user when the element's `value` is the empty string and the control is not `focused` (e.g., by displaying it inside a blank unfocused control). All U+000D CARRIAGE RETURN U+000A LINE FEED character pairs (CRLF) in the hint, as well as all other U+000D CARRIAGE RETURN (CR) and U+000A LINE FEED (LF) characters in the hint, must be treated as line breaks when rendering the hint.

The `name` attribute represents the element's name. The `dirname` attribute controls how the element's `directionality` is submitted. The `disabled` attribute is used to make the control non-interactive and to prevent its value from being submitted. The `form` attribute is used to explicitly associate the `<textarea>` element with its `form owner`. The `autofocus` attribute controls focus. The `inputmode` attribute controls the user interface's input modality for the control. The `autocomplete` attribute controls how the user agent provides autofill behavior.

This definition is non-normative. Implementation requirements are given below this definition.

`textarea` . type

Returns the string "textarea".

`textarea` . value

Returns the current value of the element.

Can be set, to change the value.

The `cols`, `placeholder`, `required`, `rows`, and `wrap` attributes must [reflect](#) the respective content attributes of the same name. The `cols` and `rows` attributes are [limited to only non-negative numbers greater than zero](#). The `cols` attribute's default value is 20. The `rows` attribute's default value is 2. The `dirname` IDL attribute must [reflect](#) the `dirname` content attribute. The `inputmode` IDL attribute must [reflect](#) the `inputmode` content attribute, [limited to only known values](#). The `maxLength` IDL attribute must [reflect](#) the `maxlength` content attribute, [limited to only non-negative numbers](#). The `minLength` IDL attribute must [reflect](#) the `minlength` content attribute, [limited to only non-negative numbers](#). The `readOnly` IDL attribute must [reflect](#) the `readonly` content attribute.

The `type` IDL attribute must return the value "textarea".

The `defaultValue` IDL attribute must act like the element's `textContent` IDL attribute.

The `value` attribute must, on getting, return the element's [API value](#); on setting, it must set the element's [raw value](#) to the new value, set the element's [dirty value flag](#) to true, and should then move the text entry cursor position to the end of the text field, unselecting any selected text and resetting the selection direction to *none*.

The `textLength` IDL attribute must return the [code-unit length](#) of the element's [API value](#).

The `willValidate`, `validity`, and `validationMessage` IDL attributes, and the `checkValidity()`, `reportValidity()`, and `setCustomValidity()` methods, are part of the [constraint validation API](#). The `labels` IDL attribute provides a list of the element's `<label>`s. The `select()`, `selectionStart`, `selectionEnd`, `selectionDirection`, `setRangeText()`, and `setSelectionRange()` methods and IDL attributes expose the element's text selection. The `autofocus`, `disabled`, `form`, and `name` IDL attributes are part of the element's forms API.

EXAMPLE 508

Here is an example of a `<textarea>` being used for unrestricted free-form text input in a form:

```
<p>If you have any comments, please let us know: <textarea cols=80 name=comments>
</textarea></p>
```

To specify a maximum length for the comments, one can use the `maxlength` attribute:

```
<p>If you have any short comments, please let us know: <textarea cols=80 name=comments
maxlength=200></textarea></p>
```

To give a default value, text can be included inside the element:

```
<p>If you have any comments, please let us know: <textarea cols=80 name=comments>You
rock!</textarea></p>
```

You can also give a minimum length. Here, a letter needs to be filled out by the user; a template (which is shorter than the minimum length) is provided, but is insufficient to submit the form:

```
<textarea required minlength="500">Dear Madam Speaker,

Regarding your letter dated ...

...

Yours Sincerely,

...</textarea>
```

A placeholder can be given as well, to suggest the basic form to the user, without providing an explicit template:

```
<textarea placeholder="Dear Francine,

They closed the parks this week, so we won't be able to
meet your there. Should we just have dinner?

Love,
Daddy"></textarea>
```

To have the browser submit [the directionality](#) of the element along with the value, the `dirname` attribute can be specified:

```
<p>If you have any comments, please let us know (you may use either English or Hebrew for your
comments):
<textarea cols=80 name=comments dirname=comments.dir></textarea></p>
```

4.10.12. The keygen element**Categories:**

[Flow content.](#)
[Phrasing content.](#)
[Interactive content.](#)
[listed](#), [labelable](#), [submittable](#), [resettable](#), and [reassociateable form-associated element](#).
[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Nothing](#).

Tag omission in text/html:

No [end tag](#).

Content attributes:

[Global attributes](#)

[autofocus](#) - Automatically focus the form control when the page is loaded

[challenge](#) - String to package with the generated and signed public key

[disabled](#) - Whether the form control is disabled

[form](#) - Associates the control with a [<form>](#) element

[keytype](#) - The type of cryptographic key to generate

[name](#) - Name of form control to use for [§4.10.22 Form submission](#) and in the [form.elements](#) API

Allowed [ARIA role attribute](#) values:

None

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

DOM interface:

```
interface HTMLKeygenElement : HTMLElement {
  attribute boolean autofocus;
  attribute DOMString challenge;
  attribute boolean disabled;
  readonly attribute HTMLFormElement? form;
  attribute DOMString keytype;
  attribute DOMString name;

  readonly attribute DOMString type;

  readonly attribute boolean willValidate;
  readonly attribute ValidityState validity;
  readonly attribute DOMString validationMessage;
  boolean checkValidity();
  boolean reportValidity();
  void setCustomValidity(DOMString error);

  [SameObject] readonly attribute NodeList labels;
};
```

This feature is in the process of being removed from the Web platform. (This is a long process that takes many years.) Using the [<keygen>](#) element at this time is highly discouraged.

The [<keygen>](#) element [represents](#) a key pair generator control. When the control's form is submitted, the private key is stored in the local keystore, and the public key is packaged and sent to the server.

The **challenge** attribute may be specified. Its value will be packaged with the submitted key.

The **keytype** attribute is an [enumerated attribute](#). The following table lists the keywords and states for the attribute — the keywords in the left column map to the states listed in the cell in the second column on the same row as the keyword. User agents are not required to support these values, and must only recognize values whose corresponding algorithms they support.

Keyword	State
rsa	<i>RSA</i>

The *invalid value default* state is the *unknown* state. The *missing value default* state is the *RSA* state, if it is supported, or the

unknown state otherwise.

NOTE:

This specification does not specify what key types user agents are to support — it is possible for a user agent to not support any key types at all.

The user agent may expose a user interface for each `<keygen>` element to allow the user to configure settings of the element's key pair generator, e.g., the key length.

The [reset algorithm](#) for `<keygen>` elements is to set these various configuration settings back to their defaults.

The element's [value](#) is the string returned from the following algorithm:

1. Use the appropriate step from the following list:

↪ If the [keytype](#) attribute is in the *RSA* state

Generate an RSA key pair using the settings given by the user, if appropriate, using the `md5WithRSASignature` RSA signature algorithm (the signature algorithm with MD5 and the RSA encryption algorithm) referenced in section 2.2.1 ("RSA Signature Algorithm") of RFC 3279, and defined in RFC 3447. [\[RFC3279\]](#) [\[RFC3447\]](#)

↪ Otherwise, the [keytype](#) attribute is in the *unknown* state

The given key type is not supported. Return the empty string and abort this algorithm.

Let *private key* be the generated private key.

Let *public key* be the generated public key.

Let *signature algorithm* be the selected signature algorithm.

2. If the element has a [challenge](#) attribute, then let *challenge* be that attribute's value. Otherwise, let *challenge* be the empty string.
3. Let *algorithm* be an ASN.1 `AlgorithmIdentifier` structure as defined by RFC 5280, with the `algorithm` field giving the ASN.1 OID used to identify *signature algorithm*, using the OIDs defined in section 2.2 ("Signature Algorithms") of RFC 3279, and the `parameters` field set up as required by RFC 3279 for `AlgorithmIdentifier` structures for that algorithm. [\[X690\]](#) [\[RFC5280\]](#) [\[RFC3279\]](#)
4. Let *spki* be an ASN.1 `SubjectPublicKeyInfo` structure as defined by RFC 5280, with the `algorithm` field set to the *algorithm* structure from the previous step, and the `subjectPublicKey` field set to the BIT STRING value resulting from ASN.1 DER encoding the *public key*. [\[X690\]](#) [\[RFC5280\]](#)
5. Let *publicKeyAndChallenge* be an ASN.1 `PublicKeyAndChallenge` structure as defined below, with the `spki` field set to the *spki* structure from the previous step, and the `challenge` field set to the string *challenge* obtained earlier. [\[X690\]](#)
6. Let *signature* be the BIT STRING value resulting from ASN.1 DER encoding the signature generated by applying the *signature algorithm* to the byte string obtained by ASN.1 DER encoding the *publicKeyAndChallenge* structure, using *private key* as the signing key. [\[X690\]](#)
7. Let *signedPublicKeyAndChallenge* be an ASN.1 `SignedPublicKeyAndChallenge` structure as defined below, with the `publicKeyAndChallenge` field set to the *publicKeyAndChallenge* structure, the `signatureAlgorithm` field set to the *algorithm* structure, and the `signature` field set to the BIT STRING *signature* from the previous step. [\[X690\]](#)
8. Return the result of base64 encoding the result of ASN.1 DER encoding the *signedPublicKeyAndChallenge* structure. [\[RFC4648\]](#) [\[X690\]](#)

The data objects used by the above algorithm are defined as follows. These definitions use the same "ASN.1-like" syntax defined by RFC 5280. [\[RFC5280\]](#)


```

PublicKeyAndChallenge ::= SEQUENCE {
    spki SubjectPublicKeyInfo,
    challenge IA5STRING
}

SignedPublicKeyAndChallenge ::= SEQUENCE {
    publicKeyAndChallenge PublicKeyAndChallenge,
    signatureAlgorithm AlgorithmIdentifier,
    signature BIT STRING
}

```



Constraint validation: The `<keygen>` element is [barred from constraint validation](#).

The [form](#) attribute is used to explicitly associate the `<keygen>` element with its [form owner](#). The [name](#) attribute represents the element's name. The [disabled](#) attribute is used to make the control non-interactive and to prevent its value from being submitted. The [autofocus](#) attribute controls focus.

This definition is non-normative. Implementation requirements are given below this definition.

***keygen* . type**

Returns the string "keygen".

The **challenge** IDL attribute must [reflect](#) the content attribute of the same name.

The **keytype** IDL attribute must [reflect](#) the content attribute of the same name, [limited to only known values](#).

The **type** IDL attribute must return the value "keygen".

The [willValidate](#), [validity](#), and [validationMessage](#) IDL attributes, and the [checkValidity\(\)](#), [reportValidity\(\)](#), and [setCustomValidity\(\)](#) methods, are part of the [constraint validation API](#). The [labels](#) IDL attribute provides a list of the element's [<label>](#)s. The [autofocus](#), [disabled](#), [form](#), and [name](#) IDL attributes are part of the element's forms API.

NOTE:

This specification does not specify how the private key generated is to be used. It is expected that after receiving the SignedPublicKeyAndChallenge (SPKAC) structure, the server will generate a client certificate and offer it back to the user for download; this certificate, once downloaded and stored in the key store along with the private key, can then be used to authenticate to services that use TLS and certificate authentication. For more information, see e.g., [this MDN article](#).

EXAMPLE 509

To generate a key pair, add the private key to the user's key store, and submit the public key to the server, markup such as the following can be used:

```

<form action="processkey.cgi" method="post" enctype="multipart/form-data">
  <p><keygen name="key"></p>
  <p><input type="submit" value="Submit key..."></p>
</form>

```

The server will then receive a [form submission](#) with a packaged RSA public key as the value of "key". This can then be used for various purposes, such as generating a client certificate, as mentioned above.

§ 4.10.13. The output element

Categories:

[Flow content.](#)

[Phrasing content.](#)

[listed](#), [labelable](#), [resettable](#), and [reassociateable](#) form-associated element.

[Palpable content](#).

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content](#).

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

[for](#) - Specifies controls from which the output was calculated

[form](#) - Associates the control with a [<form>](#) element

[name](#) - Name of form control to use for [§4.10.22 Form submission](#) and in the [form.elements](#) API

Allowed ARIA role attribute values:

[‘status’](#) (default - [do not set](#)), [Any role value](#).

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLOutputElement : HTMLElement {
  [SameObject, PutForwards=value] readonly attribute DOMTokenList htmlFor;
  readonly attribute HTMLFormElement? form;
  attribute DOMString name;

  readonly attribute DOMString type;
  attribute DOMString defaultValue;
  attribute DOMString value;

  readonly attribute boolean willValidate;
  readonly attribute ValidityState validity;
  readonly attribute DOMString validationMessage;
  boolean checkValidity();
  boolean reportValidity();
  void setCustomValidity(DOMString error);

  [SameObject] readonly attribute NodeList labels;
};
```

The [<output>](#) element [represents](#) the result of a calculation performed by the application, or the result of a user action.

NOTE:

This element can be contrasted with the [<samp>](#) element, which is the appropriate element for quoting the output of other programs run previously.

The [for](#) content attribute allows an explicit relationship to be made between the result of a calculation and the elements that represent the values that went into the calculation or that otherwise influenced the calculation. The [for](#) attribute, if specified, must contain a string consisting of an [unordered set of unique space-separated tokens](#) that are [case-sensitive](#), each of which must have the value of an [ID](#) of an element in the same [Document](#).

The [form](#) attribute is used to explicitly associate the [<output>](#) element with its [form owner](#). The [name](#) attribute represents the element's name. The [<output>](#) element is associated with a form so that it can be easily referenced from the event handlers of form controls; the element's value itself is not submitted when the form is submitted.

The element has a **value mode flag** which is either *value* or *default*. Initially, the [value mode flag](#) must be set to *default*.

The element also has a **default value**. Initially, the [default value](#) must be the empty string.

When the [value mode flag](#) is in mode *default*, the contents of the element represent both the value of the element and its [default value](#). When the [value mode flag](#) is in mode *value*, the contents of the element represent the value of the element only, and the [default value](#) is only accessible using the `defaultValue` IDL attribute.

Whenever the element's descendants are changed in any way, if the [value mode flag](#) is in mode *default*, the element's [default value](#) must be set to the value of the element's [textContent](#) IDL attribute.

The [reset algorithm](#) for `<output>` elements is to set the element's [value mode flag](#) to *default* and then to set the element's [textContent](#) IDL attribute to the value of the element's [default value](#) (thus replacing the element's child nodes).

This definition is non-normative. Implementation requirements are given below this definition.

`output . value [ = value ]`

Returns the element's current value.

Can be set, to change the value.

`output . defaultValue [ = value ]`

Returns the element's current default value.

Can be set, to change the default value.

`output . type`

Returns the string "output".

The **value** IDL attribute must act like the element's [textContent](#) IDL attribute, except that on setting, in addition, before the child nodes are changed, the element's [value mode flag](#) must be set to *value*.

The **defaultValue** IDL attribute, on getting, must return the element's [default value](#). On setting, the attribute must set the element's [default value](#), and, if the element's [value mode flag](#) is in the mode *default*, set the element's [textContent](#) IDL attribute as well.

The **type** attribute must return the string "output".

The **htmlFor** IDL attribute must [reflect](#) the [for](#) content attribute.

The [willValidate](#), [validity](#), and [validationMessage](#) IDL attributes, and the [checkValidity\(\)](#), [reportValidity\(\)](#), and [setCustomValidity\(\)](#) methods, are part of the [constraint validation API](#). The [labels](#) IDL attribute provides a list of the element's [<label>](#)s. The [form](#) and [name](#) IDL attributes are part of the element's forms API.

EXAMPLE 510

A simple calculator could use `<output>` for its display of calculated results:

```
<form onsubmit="return false" oninput="o.value = a.valueAsNumber + b.valueAsNumber">
  <input name=a type=number step=any> +
  <input name=b type=number step=any> =
  <output name=o for="a b"></output>
</form>
```

EXAMPLE 511

In this example, an `<output>` element is used to report the results of a calculation performed by a remote server, as they come in:

```
<output id="result"></output>
<script>
  var primeSource = new WebSocket('ws://primes.example.net/');
  primeSource.onmessage = function (event) {
    document.getElementById('result').value = event.data;
  }
</script>
```

§ 4.10.14. The progress element**Categories:**

[Flow content.](#)
[Phrasing content.](#)
[Labelable element.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content](#), but there must be no `<progress>` element descendants.

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)
[value](#) - Current value of the element
[max](#) - Upper bound of range

Allowed ARIA role attribute values:

`'progressbar'` (default - *do not set*).

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)
 Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLProgressElement : HTMLElement {
  attribute double value;
  attribute double max;
  readonly attribute double position;
  [SameObject] readonly attribute NodeList labels;
};
```

The `<progress>` element [represents](#) the completion progress of a task. The progress is either indeterminate, indicating that progress is being made but that it is not clear how much more work remains to be done before the task is complete (e.g., because the task is waiting for a remote host to respond), or the progress is a number in the range zero to a maximum, giving the fraction of work that has so far been completed.

There are two attributes that determine the current task completion represented by the element. The **value** content attribute specifies how much of the task has been completed, and the **max** content attribute specifies how much work the task requires in total. The units are arbitrary and not specified.

NOTE:

To make a determinate progress bar, add a [value](#) attribute with the current progress (either a number from 0.0 to 1.0, or, if the [max](#) attribute is specified, a number from 0 to the value of the [max](#) attribute). To make an indeterminate progress bar, remove the [value](#) attribute.

Authors are encouraged to also include the current value and the [maximum value](#) inline as text inside the element, so that the progress is made available to users of legacy user agents.

EXAMPLE 512

Here is a snippet of a Web application that shows the progress of some automated task:

```
<section>
  <h2>Task Progress</h2>
  <p>Progress: <progress id="p" max=100><span>0</span>%</progress></p>
  <script>
    var progressBar = document.getElementById('p');
    function updateProgress(newValue) {
      progressBar.value = newValue;
      progressBar.getElementsByTagName('span')[0].textContent = newValue;
    }
  </script>
</section>
```

(The `updateProgress()` method in this example would be called by some other code on the page to update the actual progress bar as the task progressed.)

The [value](#) and [max](#) attributes, when present, must have values that are [valid floating-point numbers](#). The [value](#) attribute, if present, must have a value equal to or greater than zero, and less than or equal to the value of the [max](#) attribute, if present, or 1.0, otherwise. The [max](#) attribute, if present, must have a value greater than zero.

NOTE:

The `<progress>` element is the wrong element to use for something that is just a gauge, as opposed to task progress. For instance, indicating disk space usage using `<progress>` would be inappropriate. Instead, the `<meter>` element is available for such use cases.

User agent requirements: If the [value](#) attribute is omitted, then the progress bar is an indeterminate progress bar. Otherwise, it is a determinate progress bar.

If the progress bar is a determinate progress bar and the element has a [max](#) attribute, the user agent must parse the [max](#) attribute's value according to the [rules for parsing floating-point number values](#). If this does not result in an error, and if the parsed value is greater than zero, then the **maximum value** of the progress bar is that value. Otherwise, if the element has no [max](#) attribute, or if it has one but parsing it resulted in an error, or if the parsed value was less than or equal to zero, then the [maximum value](#) of the progress bar is 1.0.

If the progress bar is a determinate progress bar, user agents must parse the [value](#) attribute's value according to the [rules for parsing floating-point number values](#). If this does not result in an error, and if the parsed value is less than the [maximum value](#) and greater than zero, then the **current value** of the progress bar is that parsed value. Otherwise, if the parsed value was greater than or equal to the [maximum value](#), then the [current value](#) of the progress bar is the [maximum value](#) of the progress bar. Otherwise, if parsing the [value](#) attribute's value resulted in an error, or a number less than or equal to zero, then the [current value](#) of the progress bar is zero.

user agent requirements for showing the progress bar: When representing a `<progress>` element to the user, the user agent should indicate whether it is a determinate or indeterminate progress bar, and in the former case, should indicate the relative position of the [current value](#) relative to the [maximum value](#).

This definition is non-normative. Implementation requirements are given below this definition.

`progress`.position

For a determinate progress bar (one with known current and maximum values), returns the result of dividing the

[current value](#) by the maximum value.

For an indeterminate progress bar, returns -1.

If the progress bar is an indeterminate progress bar, then the **position** IDL attribute must return -1. Otherwise, it must return the result of dividing the [current value](#) by the [maximum value](#).

If the progress bar is an indeterminate progress bar, then the **value** IDL attribute, on getting, must return 0. Otherwise, it must return the [current value](#). On setting, the given value must be converted to the [best representation of the number as a floating-point number](#) and then the [value](#) content attribute must be set to that string.

NOTE:

Setting the [value](#) IDL attribute to itself when the corresponding content attribute is absent would change the progress bar from an indeterminate progress bar to a determinate progress bar with no progress.

The **max** IDL attribute must [reflect](#) the content attribute of the same name, [limited to numbers greater than zero](#). The default value for **max** is 1.0.

The [labels](#) IDL attribute provides a list of the element's [<label>](#)s.

§ 4.10.15. The meter element

Categories:

[Flow content](#),
[Phrasing content](#),
[Labelable element](#),
[Palpable content](#).

Contexts in which this element can be used:

Where [phrasing content](#) is expected.

Content model:

[Phrasing content](#), but there must be no [<meter>](#) element descendants.

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)
value - Current value of the element
min - Lower bound of range
max - Upper bound of range
low - High limit of low range
high - Low limit of high range
optimum - Optimum value in gauge

Allowed [ARIA role attribute](#) values:

None

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

DOM interface:

```
interface HTMLMeterElement : HTMLElement {
    attribute double value;
    attribute double min;
    attribute double max;
    attribute double low;
    attribute double high;
    attribute double optimum;
    [SameObject] readonly attribute NodeList labels;
};
```

The `<meter>` element [represents](#) a scalar measurement within a known range, or a fractional value; for example disk usage, the relevance of a query result, or the fraction of a voting population to have selected a particular candidate.

This is also known as a gauge.

NOTE:

The `<meter>` element should not be used to indicate progress (as in a progress bar). For that role, HTML provides a separate `<progress>` element.

NOTE:

The `<meter>` element also does not represent a scalar value of arbitrary range — for example, it would be wrong to use this to report a weight, or height, unless there is a known [maximum value](#).

There are six attributes that determine the semantics of the gauge represented by the element.

The **min** attribute specifies the lower bound of the range, and the **max** attribute specifies the upper bound. The **value** attribute specifies the value to have the gauge indicate as the "measured" value.

The other three attributes can be used to segment the gauge's range into "low", "medium", and "high" parts, and to indicate which part of the gauge is the "optimum" part. The **low** attribute specifies the range that is considered to be the "low" part, and the **high** attribute specifies the range that is considered to be the "high" part. The **optimum** attribute gives the position that is "optimum"; if that is higher than the "high" value then this indicates that the higher the value, the better; if it's lower than the "low" mark then it indicates that lower values are better, and naturally if it is in between then it indicates that neither high nor low values are good.

Authoring requirements: The [value](#) attribute must be specified. The [value](#), [min](#), [low](#), [high](#), [max](#), and [optimum](#) attributes, when present, must have values that are [valid floating-point numbers](#).

In addition, the attributes' values are further constrained:

Let *value* be the [value](#) attribute's number.

If the [min](#) attribute is specified, then let *minimum* be that attribute's value; otherwise, let it be zero.

If the [max](#) attribute is specified, then let *maximum* be that attribute's value; otherwise, let it be 1.0.

The following inequalities must hold, as applicable:

- $minimum \leq value \leq maximum$
- $minimum \leq low \leq maximum$ (if [low](#) is specified)
- $minimum \leq high \leq maximum$ (if [high](#) is specified)
- $minimum \leq optimum \leq maximum$ (if [optimum](#) is specified)
- $low \leq high$ (if both [low](#) and [high](#) are specified)

NOTE:

If no minimum or maximum is specified, then the range is assumed to be 0..1, and the value thus has to be within that range.

Authors are encouraged to include a textual representation of the gauge's state in the element's contents, for users of user agents that do not support the `<meter>` element.

When used with microdata, the `<meter>` element's [value](#) attribute provides the element's machine-readable value.

EXAMPLE 513

The following examples show three gauges that would all be three-quarters full:

```
Storage space usage: <meter value=6 max=8>6 blocks used (out of 8 total)</meter>
Voter turnout: <meter value=0.75></meter>
Tickets sold: <meter min="0" max="100" value="75"></meter>
```

The following example is incorrect use of the element, because it doesn't give a range (and since the default maximum is 1, both of the gauges would end up looking maxed out):

```
<p>The grapefruit pie had a radius of <meter value=12>12cm</meter> and a height of <meter
value=2>2cm</meter>.</p> <!-- BAD! -->
```

Instead, one would either not include the meter element, or use the meter element with a defined range to give the dimensions in context compared to other pies:

```
<p>The grapefruit pie had a radius of 12cm and a height of
2cm.</p>
<dl>
  <dt>Radius: <dd> <meter min=0 max=20 value=12>12cm</meter>
  <dt>Height: <dd> <meter min=0 max=10 value=2>2cm</meter>
</dl>
```

There is no explicit way to specify units in the `<meter>` element, but the units may be specified in the `title` attribute in free-form text.

EXAMPLE 514

The example above could be extended to mention the units:

```
<dl>
  <dt>Radius: <dd> <meter min=0 max=20 value=12 title="centimeters">12cm</meter>
  <dt>Height: <dd> <meter min=0 max=10 value=2 title="centimeters">2cm</meter>
</dl>
```

User agent requirements: User agents must parse the `min`, `max`, `value`, `low`, `high`, and `optimum` attributes using the [rules for parsing floating-point number values](#).

User agents must then use all these numbers to obtain values for six points on the gauge, as follows. (The order in which these are evaluated is important, as some of the values refer to earlier ones.)

The minimum value

If the `min` attribute is specified and a value could be parsed out of it, then the [minimum value](#) is that value. Otherwise, the [minimum value](#) is zero.

The maximum value

If the `max` attribute is specified and a value could be parsed out of it, then the candidate [maximum value](#) is that value. Otherwise, the candidate [maximum value](#) is 1.0.

If the candidate [maximum value](#) is greater than or equal to the [minimum value](#), then the [maximum value](#) is the candidate [maximum value](#). Otherwise, the [maximum value](#) is the same as the [minimum value](#).

The actual value

If the `value` attribute is specified and a value could be parsed out of it, then that value is the candidate actual value. Otherwise, the candidate actual value is zero.

If the candidate actual value is less than the [minimum value](#), then the actual value is the [minimum value](#).

Otherwise, if the candidate actual value is greater than the [maximum value](#), then the actual value is the [maximum value](#).

Otherwise, the actual value is the candidate actual value.

The *low boundary*

If the [low](#) attribute is specified and a value could be parsed out of it, then the candidate low boundary is that value.

Otherwise, the candidate low boundary is the same as the [minimum value](#).

If the candidate low boundary is less than the [minimum value](#), then the low boundary is the [minimum value](#).

Otherwise, if the candidate low boundary is greater than the [maximum value](#), then the low boundary is the [maximum value](#).

Otherwise, the low boundary is the candidate low boundary.

The *high boundary*

If the [high](#) attribute is specified and a value could be parsed out of it, then the candidate high boundary is that value.

Otherwise, the candidate high boundary is the same as the [maximum value](#).

If the candidate high boundary is less than the low boundary, then the high boundary is the low boundary.

Otherwise, if the candidate high boundary is greater than the [maximum value](#), then the high boundary is the [maximum value](#).

Otherwise, the high boundary is the candidate high boundary.

The *optimum point*

If the [optimum](#) attribute is specified and a value could be parsed out of it, then the candidate optimum point is that value. Otherwise, the candidate optimum point is the midpoint between the [minimum value](#) and the [maximum value](#).

If the candidate optimum point is less than the [minimum value](#), then the optimum point is the [minimum value](#).

Otherwise, if the candidate optimum point is greater than the [maximum value](#), then the optimum point is the [maximum value](#).

Otherwise, the optimum point is the candidate optimum point.

All of which will result in the following inequalities all being true:

- minimum value $\leq$ actual value $\leq$ maximum value
- minimum value $\leq$ low boundary $\leq$ high boundary $\leq$ maximum value
- minimum value $\leq$ optimum point $\leq$ maximum value

user agent requirements for regions of the gauge: If the optimum point is equal to the low boundary or the high boundary, or anywhere in between them, then the region between the low and high boundaries of the gauge must be treated as the optimum region, and the low and high parts, if any, must be treated as suboptimal. Otherwise, if the optimum point is less than the low boundary, then the region between the [minimum value](#) and the low boundary must be treated as the optimum region, the region from the low boundary up to the high boundary must be treated as a suboptimal region, and the remaining region must be treated as an even less good region. Finally, if the optimum point is higher than the high boundary, then the situation is reversed; the region between the high boundary and the [maximum value](#) must be treated as the optimum region, the region from the high boundary down to the low boundary must be treated as a suboptimal region, and the remaining region must be treated as an even less good region.

user agent requirements for showing the gauge: When representing a [meter](#) element to the user, the user agent should indicate the relative position of the actual value to the minimum and maximum values, and the relationship between the actual value and the three regions of the gauge.

User agents may combine the value of the [title](#) attribute and the other attributes to provide context-sensitive help or inline text detailing the actual values.

EXAMPLE 515

For example, the following snippet:

```
<meter min=0 max=60 value=23.2 title=seconds></meter>
```

...might cause the user agent to display a gauge with a tooltip saying "Value: 23.2 out of 60." on one line and "seconds" on a second line.

The **value** IDL attribute, on getting, must return the [actual value](#). On setting, the given value must be converted to the [best representation of the number as a floating-point number](#) and then the **value** content attribute must be set to that string.

The **min** IDL attribute, on getting, must return the [minimum value](#). On setting, the given value must be converted to the [best representation of the number as a floating-point number](#) and then the **min** content attribute must be set to that string.

The **max** IDL attribute, on getting, must return the [maximum value](#). On setting, the given value must be converted to the [best representation of the number as a floating-point number](#) and then the **max** content attribute must be set to that string.

The **low** IDL attribute, on getting, must return the [low boundary](#). On setting, the given value must be converted to the [best representation of the number as a floating-point number](#) and then the **low** content attribute must be set to that string.

The **high** IDL attribute, on getting, must return the [high boundary](#). On setting, the given value must be converted to the [best representation of the number as a floating-point number](#) and then the **high** content attribute must be set to that string.

The **optimum** IDL attribute, on getting, must return the [optimum value](#). On setting, the given value must be converted to the [best representation of the number as a floating-point number](#) and then the **optimum** content attribute must be set to that string.

The **labels** IDL attribute provides a list of the element's **<label>**s.

EXAMPLE 516

The following example shows how a gauge could fall back to localized or pretty-printed text.

```
<p>Disk usage: <meter min=0 value=170261928 max=233257824>170 261 928 bytes used  
out of 233 257 824 bytes available</meter></p>
```

§ **4.10.16. The fieldset element****Categories:**

[Flow content](#).
[Sectioning root](#).
[listed](#) and [reassociateable](#) [form-associated element](#).
[Palpable content](#).

Contexts in which this element can be used:

Where [flow content](#) is expected.

Content model:

Optionally a [<legend>](#) element, followed by [flow content](#).

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)
disabled - Whether the form control is disabled
form - Associates the control with a [<form>](#) element
name - Name of form control to use for [§4.10.22 Form submission](#) and in the [form.elements](#) API

Allowed [ARIA role attribute](#) values:

`'group'` (default - *do not set*) or `'presentation'`.

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLFieldSetElement : HTMLElement {
    attribute boolean disabled;
    readonly attribute HTMLFormElement? form;
    attribute DOMString name;

    readonly attribute DOMString type;

    [SameObject] readonly attribute HTMLCollection elements;

    readonly attribute boolean willValidate;
    [SameObject] readonly attribute ValidityState validity;
    readonly attribute DOMString validationMessage;
    boolean checkValidity();
    boolean reportValidity();
    void setCustomValidity(DOMString error);
};
```

The `<fieldset>` element [represents](#) a set of form controls optionally grouped under a common name.

The name of the group is given by the first `<legend>` element that is a child of the `<fieldset>` element, if any. The remainder of the descendants form the group.

The **disabled** attribute, when specified, causes all the form control descendants of the `<fieldset>` element, excluding those that are descendants of the `<fieldset>` element's first `<legend>` element child, if any, to be disabled.

A `<fieldset>` element is a **disabled fieldset** if it matches any of the following conditions:

- Its [disabled](#) attribute is specified
- It is a descendant of another `<fieldset>` element whose [disabled](#) attribute is specified, and is *not* a descendant of that `<fieldset>` element's first `<legend>` element child, if any.

The [form](#) attribute is used to explicitly associate the `<fieldset>` element with its [form owner](#). The [name](#) attribute represents the element's name.

This definition is non-normative. Implementation requirements are given below this definition.

`fieldset` . type

Returns the string "fieldset".

`fieldset` . elements

Returns an HTMLCollection of the form controls in the element.

The **disabled** IDL attribute must [reflect](#) the content attribute of the same name.

The **type** IDL attribute must return the string "fieldset".

The **elements** IDL attribute must return an HTMLCollection rooted at the `<fieldset>` element, whose filter matches [listed elements](#).

The [willValidate](#), [validity](#), and [validationMessage](#) attributes, and the [checkValidity\(\)](#), [reportValidity\(\)](#), and [setCustomValidity\(\)](#) methods, are part of the [constraint validation API](#). The [form](#) and [name](#) IDL attributes are part of the element's forms API.

EXAMPLE 517

This example shows a `<fieldset>` element being used to group a set of related controls:

```
<fieldset>
  <legend>Display</legend>
  <p><label><input type=radio name=c value=0 checked> Black on White</label>
  <p><label><input type=radio name=c value=1> White on Black</label>
  <p><label><input type=checkbox name=g> Use grayscale</label>
  <p><label>Enhance contrast <input type=range name=e list=contrast min=0 max=100 value=0
step=1></label>
  <datalist id=contrast>
  <option label=Normal value=0>
  <option label=Maximum value=100>
  </datalist>
</fieldset>
```

EXAMPLE 518

The following snippet shows a fieldset with a checkbox in the legend that controls whether or not the fieldset is enabled. The contents of the fieldset consist of two required text fields and an optional year/month control.

```
<fieldset name="clubfields" disabled>
  <legend> <label>
    <input type=checkbox name=club onchange="form.clubfields.disabled = !checked">
    Use Club Card
  </label> </legend>
  <p><label>Name on card: <input name=clubname required></label></p>
  <p><label>Card number: <input name=clubnum required pattern="[-0-9]+"></label></p>
  <p><label>Expiry date: <input name=clubexp type=month></label></p>
</fieldset>
```

EXAMPLE 519

You can also nest `<fieldset>` elements. Here is an example expanding on the previous one that does so:

```
<fieldset name="clubfields" disabled>
  <legend> <label>
    <input type="checkbox" name="club" onchange="form.clubfields.disabled = !checked">
    Use Club Card
  </label> </legend>
  <p><label>Name on card: <input name="clubname" required></label></p>
  <fieldset name="numfields">
    <legend> <label>
      <input type="radio" checked name="clubtype" onchange="form.numfields.disabled = !checked">
      My card has numbers on it
    </label> </legend>
    <p><label>Card number: <input name="clubnum" required pattern="[-0-9]+"></label></p>
  </fieldset>
  <fieldset name="letfields" disabled>
    <legend> <label>
      <input type="radio" name="clubtype" onchange="form.letfields.disabled = !checked">
      My card has letters on it
    </label> </legend>
    <p><label>Card code: <input name="clublet" required pattern="[A-Za-z]+"></label></p>
  </fieldset>
</fieldset>
```

In this example, if the outer "Use Club Card" checkbox is not checked, everything inside the outer `<fieldset>`, including the two radio buttons in the legends of the two nested `<fieldset>`s, will be disabled. However, if the checkbox is checked, then the radio buttons will both be enabled and will let you select which of the two inner `<fieldset>`s is to be enabled.

4.10.17. The legend element

Categories:

None.

Contexts in which this element can be used:

As the first child of a `<fieldset>` element.

Content model:

Phrasing content.

Tag omission in text/html:

Neither tag is omissible

Content attributes:

Global attributes

Allowed ARIA role attribute values:

Any role value.

Allowed ARIA state and property attributes:

Global aria-* attributes

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLLegendElement : HTMLElement {
  readonly attribute HTMLFormElement? form;
};
```

The `<legend>` element [represents](#) a caption for the rest of the contents of the `<legend>` element's parent `<fieldset>` element,

if any.

This definition is non-normative. Implementation requirements are given below this definition.

`legend` . `form`

Returns the element's `<form>` element, if any, or null otherwise.

The `form` IDL attribute's behavior depends on whether the `<legend>` element is in a `<fieldset>` element or not. If the `<legend>` has a `<fieldset>` element as its parent, then the `form` IDL attribute must return the same value as the `form` IDL attribute on that `<fieldset>` element. Otherwise, it must return null.

§ 4.10.18. Form control infrastructure

§ 4.10.18.1. A form control value

Most form controls have a **value** and a **checkedness**. (The latter is only used by `<input>` elements.) These are used to describe how the user interacts with the control.

A control's **value** is its internal state. As such, it might not match the user's current input.

EXAMPLE 520

For instance, if a user enters the word "three" into a numeric field that expects digits, the user's input would be the string "three" but the control's **value** would remain unchanged. Or, if a user enters the email address " awesome@example.com" (with leading whitespace) into an **email field**, the user's input would be the string " awesome@example.com" but the browser's UI for email fields might translate that into a **value** of "awesome@example.com" (without the leading whitespace).

To define the behavior of constraint validation in the face of the `<input>` element's `multiple` attribute, `<input>` elements can also have separately defined **values**.

The `<select>` element does not have a **value**; the **selectedness** of its `<option>` elements is what is used instead.

§ 4.10.18.2. Mutability

A form control can be designated as **mutable**.

NOTE:

This determines (by means of definitions and requirements in this specification that rely on whether an element is so designated) whether or not the user can modify the **value** or **checkedness** of a form control, or whether or not a control can be automatically prefilled.

§ 4.10.18.3. Association of controls and forms

A **form-associated element** can have a relationship with a `<form>` element, which is called the element's **form owner**. If a **form-associated element** is not associated with a `<form>` element, its **form owner** is said to be null.

A **form-associated element** is, by default, associated with its nearest ancestor `<form>` element (as described below), but, if it is **reassociateable**, may have a **form** attribute specified to override this.

NOTE:

This feature allows authors to work around the lack of support for nested `<form>` elements.

If a **reassociateable form-associated element** has a **form** attribute specified, then that attribute's value must be the **ID** of a `<form>` element in the element's owner **Document**.

NOTE:

The rules in this section are complicated by the fact that although conforming documents will never contain nested `<form>` elements, it is quite possible (e.g., using a script that performs DOM manipulation) to generate documents that have such nested elements. They are also complicated by rules in the HTML parser that, for historical reasons, can result in a [form-associated element](#) being associated with a `<form>` element that is not its ancestor.

When a [form-associated element](#) is created, its [form owner](#) must be initialized to null (no owner).

When a [form-associated element](#) is to be **associated** with a form, its [form owner](#) must be set to that form.

When a [form-associated element](#) or one of its ancestors is [inserted into a Document](#), then the user agent must [reset the form owner](#) of that [form-associated element](#). The [HTML parser](#) overrides this requirement when inserting form controls.

When an element changes its parent node resulting in a [form-associated element](#) and its [form owner](#) (if any) no longer being in the same [home subtree](#), then the user agent must [reset the form owner](#) of that [form-associated element](#).

When a [reassociateable form-associated element](#)'s [form](#) attribute is set, changed, or removed, then the user agent must [reset the form owner](#) of that element.

When a [reassociateable form-associated element](#) has a [form](#) attribute and the [ID](#) of any of the elements in the [Document](#) changes, then the user agent must [reset the form owner](#) of that [form-associated element](#).

When a [reassociateable form-associated element](#) has a [form](#) attribute and an element with an [ID](#) is [inserted into](#) or [removed from](#) the [Document](#), then the user agent must [reset the form owner](#) of that [form-associated element](#).

When the user agent is to **reset the form owner** of a [form-associated element](#), it must run the following steps:

1. If the element's [form owner](#) is not null, and either the element is not [reassociateable](#) or its [form](#) content attribute is not present, and the element's [form owner](#) is its nearest `<form>` element ancestor after the change to the ancestor chain, then do nothing, and abort these steps.
2. Let the element's [form owner](#) be null.
3. If the element is [reassociateable](#), has a [form](#) content attribute, and is itself [in a Document](#), then run these substeps:
 1. If the first element [in the Document](#) to have an [ID](#) that is [case-sensitively](#) equal to the element's [form](#) content attribute's value is a `<form>` element, then associate the [form-associated element](#) with that `<form>` element.
 2. Abort the "reset the form owner" steps.
4. Otherwise, if the [form-associated element](#) in question has an ancestor `<form>` element, then associate the [form-associated element](#) with the nearest such ancestor `<form>` element.
5. Otherwise, the element is left unassociated.

EXAMPLE 521

In the following non-conforming snippet:

```
... <form id="a">
  <div id="b"></div>
</form>
<script>
document.getElementById('b').innerHTML =
  '<table><tr><td><form id="c"><input id="d"></table>' +
  '<input id="e">';
</script>
...
```

The [form owner](#) of "d" would be the inner nested form "c", while the [form owner](#) of "e" would be the outer form "a".

This happens as follows: First, the "e" node gets associated with "c" in the [HTML parser](#). Then, the innerHTML algorithm moves the nodes from the temporary document to the "b" element. At this point, the nodes see their ancestor chain change, and thus all the "magic" associations done by the parser are reset to normal ancestor associations.

This example is a non-conforming document, though, as it is a violation of the content models to nest [<form>](#) elements.

This definition is non-normative. Implementation requirements are given below this definition.

element* . **form*

Returns the element's [form owner](#).

Returns null if there isn't one.

[Reassociateable form-associated elements](#) have a **form** IDL attribute, which, on getting, must return the element's [form owner](#), or null if there isn't one.

§ 4.10.19. Attributes common to form controls

§ 4.10.19.1. Naming form controls: the *name* attribute

The **name** content attribute gives the name of the form control, as used in [§4.10.22 Form submission](#) and in the [<form>](#) element's `elements` object. If the attribute is specified, its value must not be the empty string.

Any non-empty value for [name](#) is allowed, but the name `"_charset_"` is special:

charset

This value, if used as the name of a [Hidden](#) control with no `value` attribute, is automatically given a value during submission consisting of the submission character encoding.

The **name** IDL attribute must [reflect](#) the [name](#) content attribute.

§ 4.10.19.2. Submitting element directionality: the *dirname* attribute

The **dirname** attribute on a form control element enables the submission of [the directionality](#) of the element, and gives the name of the field that contains this value during [§4.10.22 Form submission](#). If such an attribute is specified, its value must not be the empty string.

EXAMPLE 522

In this example, a form contains a text field and a submission button:

```
<form action="addcomment.cgi" method=post>
  <p><label>Comment: <input type=text name="comment" dirname="comment.dir" required>
</label></p>
  <p><button name="mode" type=submit value="add">Post Comment</button></p>
</form>
```

When the user submits the form, the user agent includes three fields, one called "comment", one called "comment.dir", and one called "mode"; so if the user types "Hello", the submission body might be something like:

```
comment=Hello&comment.dir=ltr&mode=add
```

If the user manually switches to a right-to-left writing direction and enters "مرحبا", the submission body might be something like:

```
comment=%D9%85%D8%B1%D8%AD%D8%A8%D8%A7&comment.dir=rtl&mode=add
```

4.10.19.3. Limiting user input length: the `maxLength` attribute

A **form control `maxLength` attribute**, controlled by a *dirty value flag*, declares a limit on the number of characters a user can input.

If an element has its [form control `maxLength` attribute](#) specified, the attribute's value must be a [valid non-negative integer](#). If the attribute is specified and applying the [rules for parsing non-negative integers](#) to its value results in a number, then that number is the element's **maximum allowed value length**. If the attribute is omitted or parsing its value results in an error, then there is no [maximum allowed value length](#).

Constraint validation: If an element has a [maximum allowed value length](#), its *dirty value flag* is true, its [value](#) was last changed by a user edit (as opposed to a change made by a script), and the [code-unit length](#) of the element's [value](#) is greater than the element's [maximum allowed value length](#), then the element is [suffering from being too long](#).

User agents may prevent the user from causing the element's [value](#) to be set to a value whose [code-unit length](#) is greater than the element's [maximum allowed value length](#).

NOTE:

In the case of `<textarea>` elements, this is the [value](#), not the [raw value](#), so the [textarea wrapping transformation](#) is applied before the [maximum allowed value length](#) is checked.

4.10.19.4. Setting minimum input length requirements: the `minLength` attribute

A **form control `minLength` attribute**, controlled by a *dirty value flag*, declares a lower bound on the number of characters a user can input.

NOTE:

The `minLength` attribute does not imply the `required` attribute. If the form control has no `required` attribute, then the value can still be omitted; the `minLength` attribute only kicks in once the user has entered a value at all. If the empty string is not allowed, then the `required` attribute also needs to be set.

If an element has its [form control `minLength` attribute](#) specified, the attribute's value must be a [valid non-negative integer](#). If the attribute is specified and applying the [rules for parsing non-negative integers](#) to its value results in a number, then that number is the element's **minimum allowed value length**. If the attribute is omitted or parsing its value results in an error, then there is no [minimum allowed value length](#).

If an element has both a [maximum allowed value length](#) and a [minimum allowed value length](#), the [minimum allowed value length](#) must be smaller than or equal to the [maximum allowed value length](#).

Constraint validation: If an element has a [minimum allowed value length](#), its *dirty value flag* is true, its [value](#) was last changed by a user edit (as opposed to a change made by a script), its [value](#) is not the empty string, and the [code-unit length](#) of the element's [value](#) is less than the element's [minimum allowed value length](#), then the element is [suffering from being too short](#).

EXAMPLE 523

In this example, there are four text fields. The first is required, and has to be at least 5 characters long. The other three are optional, but if the user fills one in, the user has to enter at least 10 characters.

```
<form action="/events/menu.cgi" method="post">
  <p><label>Name of Event: <input required minlength=5 maxlength=50 name=event></label></p>
  <p><label>Describe what you would like for breakfast, if anything:
    <textarea name="breakfast" minlength="10"></textarea></label></p>
  <p><label>Describe what you would like for lunch, if anything:
    <textarea name="lunch" minlength="10"></textarea></label></p>
  <p><label>Describe what you would like for dinner, if anything:
    <textarea name="dinner" minlength="10"></textarea></label></p>
  <p><input type=submit value="Submit Request"></p>
</form>
```

§ 4.10.19.5. Enabling and disabling form controls: the *disabled* attribute

The **disabled** content attribute is a [boolean attribute](#).

A form control is disabled if any of the following conditions are met:

1. The element is a [button](#), [input](#), [select](#), or [textarea](#) element, and the `disabled` attribute is specified on this element (regardless of its value).
2. The element is a descendant of a [fieldset](#) element whose `disabled` attribute is specified, and is *not* a descendant of that [fieldset](#) element's first [legend](#) element child, if any.

A form control that is disabled must prevent any [click](#) events that are [queued](#) on the [user interaction task source](#) from being dispatched on the element.

Constraint validation: If an element is disabled, it is [barred from constraint validation](#).

The **disabled** IDL attribute must [reflect](#) the [disabled](#) content attribute.

§ 4.10.19.6. Form submission

Attributes for form submission can be specified both on [form](#) elements and on [submit buttons](#) (elements that represent buttons that submit forms, e.g., an [input](#) element whose `type` attribute is in the [submit button](#) state).

The [attributes for form submission](#) that may be specified on [form](#) elements are [action](#), [enctype](#), [method](#), [novalidate](#), and [target](#).

The corresponding [attributes for form submission](#) that may be specified on [submit buttons](#) are [formaction](#), [formenctype](#), [formmethod](#), [formnovalidate](#), and [formtarget](#). When omitted, they default to the values given on the corresponding attributes on the [form](#) element.



The **action** and **formaction** content attributes, if specified, must have a value that is a [valid non-empty URL potentially surrounded by spaces](#).

The **action** of an element is the value of the element's [formaction](#) attribute, if the element is a [submit button](#) and has such an attribute, or the value of its [form owner's](#) [action](#) attribute, if *it* has one, or else the empty string.



The **method** and **formmethod** content attributes are [enumerated attributes](#) with the following keywords and states:

- The keyword **get**, mapping to the state **GET**, indicating the HTTP GET method.
- The keyword **post**, mapping to the state **POST**, indicating the HTTP POST method.

The *invalid value default* for these attributes is the [GET](#) state. The *missing value default* for the [method](#) attribute is also the [GET](#) state. (There is no *missing value default* for the [formmethod](#) attribute.)

The **method** of an element is one of those states. If the element is a [submit button](#) and has a [formmethod](#) attribute, then the element's [method](#) is that attribute's state; otherwise, it is the [form owner's method](#) attribute's state.

EXAMPLE 524

Here the [method](#) attribute is used to explicitly specify the default value, "[get](#)", so that the search query is submitted in the URL:

```
<form method="get" action="/search.cgi">
  <p><label>Search terms: <input type=search name=q></label></p>
  <p><input type=submit></p>
</form>
```

EXAMPLE 525

On the other hand, here the [method](#) attribute is used to specify the value "[post](#)", so that the user's message is submitted in the HTTP request's body:

```
<form method="post" action="/post-message.cgi">
  <p><label>Message: <input type=text name=m></label></p>
  <p><input type=submit value="Submit message"></p>
</form>
```



The **enctype** and **formenctype** content attributes are [enumerated attributes](#) with the following keywords and states:

- The "**application/x-www-form-urlencoded**" keyword and corresponding state.
- The "**multipart/form-data**" keyword and corresponding state.
- The "**text/plain**" keyword and corresponding state.

The *invalid value default* for these attributes is the [application/x-www-form-urlencoded](#) state. The *missing value default* for the [enctype](#) attribute is also the [application/x-www-form-urlencoded](#) state. (There is no *missing value default* for the [formenctype](#) attribute.)

The **enctype** of an element is one of those three states. If the element is a [submit button](#) and has a [formenctype](#) attribute, then the element's [enctype](#) is that attribute's state; otherwise, it is the [form owner's enctype](#) attribute's state.



The **target** and **formtarget** content attributes, if specified, must have values that are [valid browsing context names or keywords](#).

The [target](#) of an element is the value of the element's [formtarget](#) attribute, if the element is a [submit button](#) and has such an attribute; or the value of its [form owner's target](#) attribute, if it has such an attribute; or, if the [Document](#) contains a [<base>](#) element with a [target](#) attribute, then the value of the [target](#) attribute of the first such [<base>](#) element; or, if there is no such element, the empty string.



The **novalidate** and **formnovalidate** content attributes are [boolean attributes](#). If present, they indicate that the form is not to be validated during submission.

The **no-validate state** of an element is true if the element is a [submit button](#) and the element's [formnovalidate](#) attribute is present, or if the element's [form owner's novalidate](#) attribute is present, and false otherwise.

EXAMPLE 526

This attribute is useful to include "save" buttons on forms that have validation constraints, to allow users to save their progress even though they haven't fully entered the data in the form. The following example shows a simple form that has two required fields. There are three buttons: one to submit the form, which requires both fields to be filled in; one to save the form so that the user can come back and fill it in later; and one to cancel the form altogether.

```
<form action="editor.cgi" method="post">
  <p><label>Name: <input required name=fn></label></p>
  <p><label>Essay: <textarea required name=essay></textarea></label></p>
  <p><input type=submit name=submit value="Submit essay"></p>
  <p><input type=submit formnovalidate name=save value="Save essay"></p>
  <p><input type=submit formnovalidate name=cancel value="Cancel"></p>
</form>
```



The **action** IDL attribute must [reflect](#) the content attribute of the same name, except that on getting, when the content attribute is missing or its value is the empty string, [the document's address](#) must be returned instead.

The **target** IDL attribute must [reflect](#) the content attribute of the same name.

The **method** and **enctype** IDL attributes must [reflect](#) the respective content attributes of the same name, [limited to only known values](#).

The **encoding** IDL attribute must [reflect](#) the [enctype](#) content attribute, [limited to only known values](#).

The **noValidate** IDL attribute must [reflect](#) the [novalidate](#) content attribute.

The **formAction** IDL attribute must [reflect](#) the [formaction](#) content attribute, except that on getting, when the content attribute is missing or its value is the empty string, [the document's address](#) must be returned instead.

The **formEnctype** IDL attribute must [reflect](#) the [formenctype](#) content attribute, [limited to only known values](#).

The **formMethod** IDL attribute must [reflect](#) the [formmethod](#) content attribute, [limited to only known values](#).

The **formNoValidate** IDL attribute must [reflect](#) the [formnovalidate](#) content attribute.

The **formTarget** IDL attribute must [reflect](#) the [formtarget](#) content attribute.

§ 4.10.19.6.1. AUTOFOCUSING A FORM CONTROL: THE autofocus ATTRIBUTE

The **autofocus** content attribute allows the author to indicate that a control is to be focused as soon as the page is loaded, allowing the user to just start typing without having to manually focus the main control.

The [autofocus](#) attribute is a [boolean attribute](#).

An element's **nearest ancestor autofocus scoping root element** is the element itself if it is the element's [root element](#).

There must not be two elements with the same [nearest ancestor autofocus scoping root element](#) that both have the [autofocus](#) attribute specified.

When an element with the [autofocus](#) attribute specified is [inserted into a document](#), user agents should run the following steps:

1. Let *target* be the element's [node document](#).
2. If *target* has no [browsing context](#), abort these steps.
3. If *target*'s [browsing context](#) has no [top-level browsing context](#) (e.g., it is a [nested browsing context](#) with no [parent browsing context](#)), abort these steps.
4. If *target*'s [active sandboxing flag set](#) has the [sandboxed automatic features browsing context flag](#), abort these steps.
5. If *target*'s [origin](#) is not the same as the [origin](#) of the [node document](#) of the currently focused element in *target*'s [top-level browsing context](#), abort these steps.
6. If *target*'s [origin](#) is not the same as the [origin](#) of the [active document](#) of *target*'s [top-level browsing context](#), abort these steps.
7. If the user agent has already reached the last step of this list of steps in response to an element being [inserted](#) into a Document whose [top-level browsing context](#)'s [active document](#) is the same as *target*'s [top-level browsing context](#)'s [active document](#), abort these steps.
8. If the user has indicated (for example, by starting to type in a form control) that he does not wish focus to be changed, then optionally abort these steps.
9. [Queue a task](#) that runs the [focusing steps](#) for the element. User agents may also change the scrolling position of the document, or perform some other action that brings the element to the user's attention. The [task source](#) for this task is the [user interaction task source](#).

NOTE:

Focusing the control does not imply that the user agent must focus the browser window if it has lost focus.

The **autofocus** IDL attribute must [reflect](#) the content attribute of the same name.

EXAMPLE 527

In the following snippet, the text control would be focused when the document was loaded.

```
<input maxlength="256" name="q" value="" autofocus>


```

4.10.19.7. Input modalities: the *inputmode* attribute

The **inputmode** content attribute is an [enumerated attribute](#) that specifies what kind of input mechanism would be most helpful for users entering content into the form control.

User agents must recognize all the keywords and corresponding states given below, but need not support all of the corresponding states. If a keyword's state is not supported, the user agent must act as if the keyword instead mapped to the given state's fallback state, as defined below. This fallback behavior is transitive.

NOTE:

For example, if a user agent with a QWERTY keyboard layout does not support text prediction and automatic capitalization, then it could treat the *latin-prose* keyword in the same way as the *verbatim* keyword, following the chain [Latin Prose](#) → [Latin Text](#) → [Latin Verbatim](#).

The possible keywords and states for the attributes are listed in the following table. The keywords are listed in the first column. Each maps to the state given in the cell in the second column of that keyword's row, and that state has the fallback state given in the cell in the third column of that row.

Keyword	State	Fallback state	Description
' verbatim '	Latin Verbatim	Default	Alphanumeric Latin-script input of non-prose content, e.g., usernames, passwords, product codes.

Keyword	State	Fallback state	Description
‘latin’	Latin Text	Latin Verbatim	Latin-script input in the user’s preferred language(s), with some typing aids enabled (e.g., text prediction). Intended for human-to-computer communications, e.g., free-form text search fields.
‘latin-name’	Latin Name	Latin Text	Latin-script input in the user’s preferred language(s), with typing aids intended for entering human names enabled (e.g., text prediction from the user’s contact list and automatic capitalization at every word). Intended for situations such as customer name fields.
‘latin-prose’	Latin Prose	Latin Text	Latin-script input in the user’s preferred language(s), with aggressive typing aids intended for human-to-human communications enabled (e.g., text prediction and automatic capitalization at the start of sentences). Intended for situations such as e-mails and instant messaging.
‘full-width-latin’	Full-width Latin	Latin Prose	Latin-script input in the user’s secondary language(s), using full-width characters, with aggressive typing aids intended for human-to-human communications enabled (e.g., text prediction and automatic capitalization at the start of sentences). Intended for latin text embedded inside CJK text.
‘kana’	Kana	Default	Kana or romaji input, typically hiragana input, using full-width characters, with support for converting to kanji. Intended for Japanese text input.
‘kana-name’	Kana Name	Kana	Kana or romaji input, typically hiragana input, using full-width characters, with support for converting to kanji, and with typing aids intended for entering human names enabled (e.g., text prediction from the user’s contact list). Intended for situations such as customer name fields.
‘katakana’	Katakana	Kana	Katakana input, using full-width characters, with support for converting to kanji. Intended for Japanese text input.
‘numeric’	Numeric	Default	Numeric input, including keys for the digits 0 to 9, the user’s preferred thousands separator character, and the character for indicating negative numbers. Intended for numeric codes, e.g., credit card numbers. (For numbers, prefer "<input type=numbers>".)
‘tel’	Telephone	Numeric	Telephone number input, including keys for the digits 0 to 9, the "#" character, and the "*" character. In some locales, this can also include alphabetic mnemonic labels (e.g., in the US, the key labeled "2" is historically also labeled with the letters A, B, and C). Rarely necessary ; use "<input type=tel>" instead.
‘email’	E-mail	Default	Text input in the user’s locale, with keys for aiding in the input of e-mail addresses, such as that for the "@" character and the "." character. Rarely necessary ; use "<input type=email>" instead.
‘url’	URL	Default	Text input in the user’s locale, with keys for aiding in the input of Web addresses, such as that for the "/" and "." characters and for quick input of strings commonly found in domain names such as "www." or ".co.uk". Rarely necessary ; use "<input type=url>" instead.

The last three keywords listed above are only provided for completeness, and are **rarely necessary**, as dedicated input controls exist for their usual use cases (as described in the table above).

User agents must all support the **Default** input mode state, which corresponds to the user agent’s default input modality. This specification does not define how the user agent’s default modality is to operate. The *missing value default* is the [Default](#) input mode state.

User agents should use the input modality corresponding to the state of the `inputmode` attribute when exposing a user interface for editing the value of a form control to which the attribute [applies](#). An input modality corresponding to a state is one designed to fit the description of the state in the table above. This value can change dynamically; user agents should update their interface as the attribute changes state, unless that would go against the user’s wishes.

§ 4.10.19.8.1. AUTOFILLING FORM CONTROLS: THE `autocomplete` ATTRIBUTE

User agents sometimes have features for helping users fill forms in, for example prefilling the user's address based on earlier user input. The **`autocomplete`** content attribute can be used to hint to the user agent how to, or indeed whether to, provide such a feature.

There are two ways this attribute is used. When wearing the **autofill expectation mantle**, the [`autocomplete`](#) attribute describes what input is expected from users. When wearing the **autofill anchor mantle**, the [`autocomplete`](#) attribute describes the meaning of the given value.

On an `<input>` element whose [`type`](#) attribute is in the [Hidden](#) state, the [`autocomplete`](#) attribute wears the [autofill anchor mantle](#). In all other cases, it wears the [autofill expectation mantle](#).

When wearing the [autofill expectation mantle](#), the [`autocomplete`](#) attribute, if specified, must have a value that is an ordered [set of space-separated tokens](#) consisting of either a single token that is an [ASCII case-insensitive](#) match for the string "off", or a single token that is an [ASCII case-insensitive](#) match for the string "on", or [autofill detail tokens](#).

When wearing the [autofill anchor mantle](#), the [`autocomplete`](#) attribute, if specified, must have a value that is an ordered [set of space-separated tokens](#) consisting of just [autofill detail tokens](#) (i.e., the "on" and "off" keywords are not allowed).

Autofill detail tokens are the following, in the order given below:

1. Optionally, a token whose first eight characters are an [ASCII case-insensitive](#) match for the string "section-", meaning that the field belongs to the named group.

EXAMPLE 528

For example, if there are two shipping addresses in the form, then they could be marked up as:

```
<fieldset>
  <legend>Ship the blue gift to...</legend>
  <p> <label> Address:    <input name=ba autocomplete="section-blue shipping street-
address"> </label>
  <p> <label> City:      <input name=bc autocomplete="section-blue shipping address-
level2"> </label>
  <p> <label> Postal Code: <input name=bp autocomplete="section-blue shipping postal-
code"> </label>
</fieldset>
<fieldset>
  <legend>Ship the red gift to...</legend>
  <p> <label> Address:    <input name=ra autocomplete="section-red shipping street-
address"> </label>
  <p> <label> City:      <input name=rc autocomplete="section-red shipping address-
level2"> </label>
  <p> <label> Postal Code: <input name=rp autocomplete="section-red shipping postal-code">
</label>
</fieldset>
```

2. Optionally, a token that is an [ASCII case-insensitive](#) match for one of the following strings:

- "shipping", meaning the field is part of the shipping address or contact information
- "billing", meaning the field is part of the billing address or contact information

3. Either of the following two options:

- A token that is an [ASCII case-insensitive](#) match for one of the following [autofill field](#) names, excluding those that are [inappropriate for the control](#):

- "name"
- "honorific-prefix"
- "given-name"
- "additional-name"
- "family-name"
- "honorific-suffix"
- "nickname"
- "username"
- "new-password"

- "current-password"
- "organization-title"
- "organization"
- "street-address"
- "address-line1"
- "address-line2"
- "address-line3"
- "address-level4"
- "address-level3"
- "address-level2"
- "address-level1"
- "country"
- "country-name"
- "postal-code"
- "cc-name"
- "cc-given-name"
- "cc-additional-name"
- "cc-family-name"
- "cc-number"
- "cc-exp"
- "cc-exp-month"
- "cc-exp-year"
- "cc-csc"
- "cc-type"
- "transaction-currency"
- "transaction-amount"
- "language"
- "bday"
- "bday-day"
- "bday-month"
- "bday-year"
- "sex"
- "url"
- "photo"

(See the table below for descriptions of these values.)

- The following, in the given order:

1. Optionally, a token that is an [ASCII case-insensitive](#) match for one of the following strings:

- "home", meaning the field is for contacting someone at their residence
- "work", meaning the field is for contacting someone at their workplace
- "mobile", meaning the field is for contacting someone regardless of location
- "fax", meaning the field describes a fax machine's contact details
- "pager", meaning the field describes a pager's or beeper's contact details

2. A token that is an [ASCII case-insensitive](#) match for one of the following [autofill field](#) names, excluding those that are [inappropriate for the control](#):

- "tel"
- "tel-country-code"
- "tel-national"
- "tel-area-code"
- "tel-local"
- "tel-local-prefix"
- "tel-local-suffix"
- "tel-extension"
- "email"
- "imp"

(See the table below for descriptions of these values.)

As noted earlier, the meaning of the attribute and its keywords depends on the mantle that the attribute is wearing.

↪ When wearing the [autofill expectation mantle](#)...

The **"off"** keyword indicates either that the control's input data is particularly sensitive (for example the activation code for a nuclear weapon); or that it is a value that will never be reused (for example a one-time-key for a bank login) and the user will therefore have to explicitly enter the data each time, instead of being able to rely on the user agent to prefill the value for him; or that the document provides its own autocomplete mechanism and does not want the user agent to provide autocompletion values.

The **"on"** keyword indicates that the user agent is allowed to provide the user with autocompletion values, but does not provide any further information about what kind of data the user might be expected to enter. User agents

would have to use heuristics to decide what autocompletion values to suggest.

The [autofill field](#) listed above indicate that the user agent is allowed to provide the user with autocompletion values, and specifies what kind of value is expected. The meaning of each such keyword is described in the table below.

If the [autocomplete](#) attribute is omitted, the default value corresponding to the state of the element's [form owner's autocomplete](#) attribute is used instead (either "on" or "off"). If there is no [form owner](#), then the value "on" is used.

↪ When wearing the [autofill anchor mantle](#)...

The [autofill field](#) listed above indicate that the value of the particular kind of value specified is that value provided for this element. The meaning of each such keyword is described in the table below.

EXAMPLE 529

In this example the page has explicitly specified the currency and amount of the transaction. The form requests a credit card and other billing details. The user agent could use this information to suggest a credit card that it knows has sufficient balance and that supports the relevant currency.

```
<form method=post action="step2.cgi">
  <input type=hidden autocomplete=transaction-currency value="CHF">
  <input type=hidden autocomplete=transaction-amount value="15.00">
  <p><label>Credit card number: <input type=text inputmode=numeric autocomplete=cc-
number></label>
  <p><label>Expiry Date: <input type=month autocomplete=cc-exp></label>
  <p><input type=submit value="Continue...">
</form>
```

The **autofill field** keywords relate to each other as described in the table below. Each field name listed on a row of this table corresponds to the meaning given in the cell for that row in the column labeled "Meaning". Some fields correspond to sub-parts of other fields; for example, a credit card expiry date can be expressed as one field giving both the month and year of expiry ("cc-exp"), or as two fields, one giving the month ("cc-exp-month") and one the year ("cc-exp-year"). In such cases, the names of the broader fields cover multiple rows, in which the narrower fields are defined.

NOTE:

Generally, authors are encouraged to use the broader fields rather than the narrower fields, as the narrower fields tend to expose Western biases. For example, while it is common in some Western cultures to have a given name and a family name, in that order (and thus often referred to as a first name and a surname), many cultures put the family name first and the given name second, and many others simply have one name (a mononym). Having a single field is therefore more flexible.

Some fields are only appropriate for certain form controls. An [autofill field](#) name is **inappropriate for a control** if the control does not belong to the group listed for that [autofill field](#) in the fifth column of the first row describing that [autofill field](#) in the table below. What controls fall into each group is described below the table.

Field name	Meaning	Canonical Format	Canonical Format Example	Control group
"name"	Full name	Free-form text, no newlines	Sir Timothy John Berners-Lee, OM, KBE, FRS, FEng, FRSA	Text
"honorific-prefix"	Prefix or title (e.g., "Mr.", "Ms.", "Dr.", "M ^{lle} ")	Free-form text, no newlines	Sir	Text
"given-name"	Given name (in some Western cultures, also known as the <i>first name</i>)	Free-form text, no newlines	Timothy	Text
"additional-name"	Additional names (in some Western cultures, also known as <i>middle names</i> , forenames other	Free-form text, no newlines	John	Text

Field name	Meaning	Canonical Format	Canonical Format Example	Control group
	than the first name)			
"family-name"	Family name (in some Western cultures, also known as the <i>last name</i> or <i>surname</i>)	Free-form text, no newlines	Berners-Lee	Text
"honorific-suffix"	Suffix (e.g., "Jr.", "B.Sc.", "MBASW", "II")	Free-form text, no newlines	OM, KBE, FRS, FREng, FRSA	Text
"nickname"	Nickname, screen name, handle: a typically short name used instead of the full name	Free-form text, no newlines	Tim	Text
"organization-title"	Job title (e.g., "Software Engineer", "Senior Vice President", "Deputy Managing Director")	Free-form text, no newlines	Professor	Text
"username"	A username	Free-form text, no newlines	timbl	Text
"new-password"	A new password (e.g., when creating an account or changing a password)	Free-form text, no newlines	GUMFXbadyrS3	Password
"current-password"	The current password for the account identified by the username field (e.g., when logging in)	Free-form text, no newlines	qwerty	Password
"organization"	Company name corresponding to the person, address, or contact information in the other fields associated with this field	Free-form text, no newlines	World Wide Web Consortium	Text
"street-address"	Street address (multiple lines, newlines preserved)	Free-form text	32 Vassar Street MIT Room 32-G524	<Multiline>
"address-line1"	Street address (one line per field)	Free-form text, no newlines	32 Vassar Street	Text
"address-line2"		Free-form text, no newlines	MIT Room 32-G524	Text
"address-line3"		Free-form text, no newlines		Text
"address-level4"	The most fine-grained administrative level , in addresses with four administrative levels	Free-form text, no newlines		Text
"address-level3"	The third administrative level , in addresses with three or more administrative levels	Free-form text, no newlines		Text
"address-level2"	The second administrative level , in addresses with two or more administrative levels; in the countries with two administrative levels, this would typically be the city, town, village, or other locality within which the relevant street address is found	Free-form text, no newlines	Cambridge	Text
"address-level1"	The broadest administrative level in the address, i.e., the province within which the locality is found; for example, in the US, this would be the state; in Switzerland it would be the canton; in the UK, the post town	Free-form text, no newlines	MA	Text
"country"	Country code	Valid ISO 3166-1-alpha-2 country code [ISO3166]	US	Text
"country-name"	Country name	Free-form text, no newlines; derived from country in some cases	US	Text

Field name	Meaning	Canonical Format	Canonical Format Example	Control group
"postal-code"	Postal code, post code, ZIP code, CEDEX code (if CEDEX, append "CEDEX", and the <i>dissement</i> , if relevant, to the <code>address-level2</code> field)	Free-form text, no newlines	02139	Text
"cc-name"	Full name as given on the payment instrument	Free-form text, no newlines	Tim Berners-Lee	Text
"cc-given-name"	Given name as given on the payment instrument (in some Western cultures, also known as the <i>first name</i>)	Free-form text, no newlines	Tim	Text
"cc-additional-name"	Additional names given on the payment instrument (in some Western cultures, also known as <i>middle names</i> , forenames other than the first name)	Free-form text, no newlines		Text
"cc-family-name"	Family name given on the payment instrument (in some Western cultures, also known as the <i>last name</i> or <i>surname</i>)	Free-form text, no newlines	Berners-Lee	Text
"cc-number"	Code identifying the payment instrument (e.g., the credit card number)	ASCII digits	4114360123456785	Text
"cc-exp"	Expiration date of the payment instrument	Valid month string	2014-12	Month
"cc-exp-month"	Month component of the expiration date of the payment instrument	valid integer in the range 1..12	12	Numeric
"cc-exp-year"	Year component of the expiration date of the payment instrument	valid integer greater than zero	2014	Numeric
"cc-csc"	Security code for the payment instrument (also known as the card security code (CSC), card validation code (CVC), card verification value (CVV), signature panel code (SPC), credit card ID (CCID), etc)	ASCII digits	419	Text
"cc-type"	Type of payment instrument	Free-form text, no newlines	Visa	Text
"transaction-currency"	The currency that the user would prefer the transaction to use	ISO 4217 currency code [ISO4217]	GBP	Text
"transaction-amount"	The amount that the user would like for the transaction (e.g., when entering a bid or sale price)	Valid floating-point number	401.00	Numeric
"language"	Preferred language	Valid BCP 47 language tag [BCP47]	en	Text
"bday"	Birthday	Valid date string	1955-06-08	Date
"bday-day"	Day component of birthday	valid integer in the range 1..31	8	Numeric
"bday-month"	Month component of birthday	valid integer in the range 1..12	6	Numeric
"bday-year"	Year component of birthday	valid integer greater than zero	1955	Numeric

Field name	Meaning	Canonical Format	Canonical Format Example	Control group
"sex"	Gender identity (e.g., Female, Fa'afafine)	Free-form text, no newlines	Male	Text
"url"	Home page or other Web page corresponding to the company, person, address, or contact information in the other fields associated with this field	Valid URL	https://www.w3.org/People/Berners-Lee/	URL
"photo"	Photograph, icon, or other image corresponding to the company, person, address, or contact information in the other fields associated with this field	Valid URL	https://www.w3.org/Press/Stock/Berners-Lee/2001-europaeum-eighth.jpg	URL
"tel"	Full telephone number, including country code	ASCII digits and U+0020 SPACE characters, prefixed by a U+002B PLUS SIGN character (+)	+1 617 253 5702	Tel
"tel-country-code"	Country code component of the telephone number	ASCII digits prefixed by a U+002B PLUS SIGN character (+)	+1	Text
"tel-national"	Telephone number without the country code component, with a country-internal prefix applied if applicable	ASCII digits and U+0020 SPACE characters	617 253 5702	Text
"tel-area-code"	Area code component of the telephone number, with a country-internal prefix applied if applicable	ASCII digits	617	Text
"tel-local"	Telephone number without the country code and area code components	ASCII digits	2535702	Text
"tel-local-prefix"	First part of the component of the telephone number that follows the area code, when that component is split into two components	ASCII digits	253	Text
"tel-local-suffix"	Second part of the component of the telephone number that follows the area code, when that component is split into two components	ASCII digits	5702	Text
"tel-extension"	Telephone number internal extension code	ASCII digits	1000	Text
"email"	E-mail address	Valid e-mail address	timbl@w3.org	E-mail
"impp"	URL representing an instant messaging protocol endpoint (for example, "aim:goim?screenname=example" or "xmpp:fred@example.net")	Valid URL	irc://example.org/timbl,isuser	URL

The groups correspond to controls as follows:

Text

[<input>](#) elements with a [type](#) attribute in the [Hidden](#) state

[<input>](#) elements with a [type](#) attribute in the [Text](#) state

[<input>](#) elements with a [type](#) attribute in the [Search](#) state

[<textarea>](#) elements

[<select>](#) elements

Multiline

[<input>](#) elements with a [type](#) attribute in the [Hidden](#) state

[<textarea>](#) elements

[<select>](#) elements

Password

[<input>](#) elements with a [type](#) attribute in the [Hidden](#) state

[<input>](#) elements with a [type](#) attribute in the [Text](#) state

[<input>](#) elements with a [type](#) attribute in the [Search](#) state

[<input>](#) elements with a [type](#) attribute in the [Password](#) state

[<textarea>](#) elements

[<select>](#) elements

URL

[<input>](#) elements with a [type](#) attribute in the [Hidden](#) state

[<input>](#) elements with a [type](#) attribute in the [Text](#) state

[<input>](#) elements with a [type](#) attribute in the [Search](#) state

[<input>](#) elements with a [type](#) attribute in the [URL](#) state

[<textarea>](#) elements

[<select>](#) elements

E-mail

[<input>](#) elements with a [type](#) attribute in the [Hidden](#) state

[<input>](#) elements with a [type](#) attribute in the [Text](#) state

[<input>](#) elements with a [type](#) attribute in the [Search](#) state

[<input>](#) elements with a [type](#) attribute in the [E-mail](#) state

[<textarea>](#) elements

[<select>](#) elements

Tel

[<input>](#) elements with a [type](#) attribute in the [Hidden](#) state

[<input>](#) elements with a [type](#) attribute in the [Text](#) state

[<input>](#) elements with a [type](#) attribute in the [Search](#) state

[<input>](#) elements with a [type](#) attribute in the [Telephone](#) state

[<textarea>](#) elements

[<select>](#) elements

Numeric

[<input>](#) elements with a [type](#) attribute in the [Hidden](#) state

[<input>](#) elements with a [type](#) attribute in the [Text](#) state

[<input>](#) elements with a [type](#) attribute in the [Search](#) state

[<input>](#) elements with a [type](#) attribute in the [Number](#) state

[<textarea>](#) elements

[<select>](#) elements

Month

[<input>](#) elements with a [type](#) attribute in the [Hidden](#) state

[<input>](#) elements with a [type](#) attribute in the [Text](#) state

[<input>](#) elements with a [type](#) attribute in the [Search](#) state

[<input>](#) elements with a [type](#) attribute in the [Month](#) state

[<textarea>](#) elements

[<select>](#) elements

Date

[<input>](#) elements with a [type](#) attribute in the [Hidden](#) state

[<input>](#) elements with a [type](#) attribute in the [Text](#) state

[<input>](#) elements with a [type](#) attribute in the [Search](#) state

[<input>](#) elements with a [type](#) attribute in the [Date](#) state

[<textarea>](#) elements

[<select>](#) elements

Address levels: The "address-level1" – "address-level4" fields are used to describe the locality of the street address. Different locales have different numbers of levels. For example, the US uses two levels (state and town), the UK uses one or two depending on the address (the post town, and in some cases the locality), and China can use three (province, city, district). The "address-level1" field represents the widest administrative division. Different locales order the fields in different ways; for example, in the US the town (level 2) precedes the state (level 1); while in Japan the prefecture (level 1) precedes the city (level 2) which precedes the district (level 3). Authors are encouraged to provide forms that are presented in a way that matches the country's conventions (hiding, showing, and rearranging fields accordingly as the user changes the country).

§ 4.10.19.8.2. PROCESSING MODEL

Each [<input>](#) element to which the [autocomplete](#) attribute applies, each [<select>](#) element, and each [<textarea>](#) element, has an **autofill hint set**, an **autofill scope**, an **autofill field name**, and an **IDL-exposed autofill value**.

The [autofill field name](#) specifies the specific kind of data expected in the field, e.g., "street-address" or "cc-exp".

The [autofill hint set](#) identifies what address or contact information type the user agent is to look at, e.g., "shipping fax" or "billing".

The [autofill scope](#) identifies the group of fields that are to be filled with the information from the same source, and consists of the [autofill hint set](#) with, if applicable, the "section-*" prefix, e.g., "billing", "section-parent shipping", or "section-child shipping home".

These values are defined as the result of running the following algorithm:

1. If the element has no [autocomplete](#) attribute, then jump to the step labeled *default*.
2. Let *tokens* be the result of [splitting the attribute's value on spaces](#).
3. If *tokens* is empty, then jump to the step labeled *default*.
4. Let *index* be the index of the last token in *tokens*.
5. If the *index*th token in *tokens* is not an [ASCII case-insensitive](#) match for one of the tokens given in the first column of the following table, or if the number of tokens in *tokens* is greater than the maximum number given in the cell in the second column of that token's row, then jump to the step labeled *default*. Otherwise, let *field* be the string given in the cell of the first column of the matching row, and let *category* be the value of the cell in the third column of that same row.

Token	Maximum number of tokens	Category
"off"	1	Off
"on"	1	Automatic
"name"	3	Normal

Token	Maximum number of tokens	Category
"honorific-prefix"	3	Normal
"given-name"	3	Normal
"additional-name"	3	Normal
"family-name"	3	Normal
"honorific-suffix"	3	Normal
"nickname"	3	Normal
"organization-title"	3	Normal
"username"	3	Normal
"new-password"	3	Normal
"current-password"	3	Normal
"organization"	3	Normal
"street-address"	3	Normal
"address-line1"	3	Normal
"address-line2"	3	Normal
"address-line3"	3	Normal
"address-level4"	3	Normal
"address-level3"	3	Normal
"address-level2"	3	Normal
"address-level1"	3	Normal
"country"	3	Normal
"country-name"	3	Normal
"postal-code"	3	Normal
"cc-name"	3	Normal
"cc-given-name"	3	Normal
"cc-additional-name"	3	Normal
"cc-family-name"	3	Normal
"cc-number"	3	Normal
"cc-exp"	3	Normal
"cc-exp-month"	3	Normal
"cc-exp-year"	3	Normal
"cc-csc"	3	Normal
"cc-type"	3	Normal
"transaction-currency"	3	Normal
"transaction-amount"	3	Normal
"language"	3	Normal
"bday"	3	Normal
"bday-day"	3	Normal
"bday-month"	3	Normal
"bday-year"	3	Normal
"sex"	3	Normal
"url"	3	Normal
"photo"	3	Normal
"tel"	4	Contact

Token	Maximum number of tokens	Category
"tel-country-code"	4	Contact
"tel-national"	4	Contact
"tel-area-code"	4	Contact
"tel-local"	4	Contact
"tel-local-prefix"	4	Contact
"tel-local-suffix"	4	Contact
"tel-extension"	4	Contact
"email"	4	Contact
"impp"	4	Contact

6. If `category` is Off or Automatic but the element's `autocomplete` attribute is wearing the `autofill anchor mantle`, then jump to the step labeled *default*.
7. If `category` is Off, let the element's `autofill field name` be the string "off", let its `autofill hint set` be empty, and let its `IDL-exposed autofill value` be the string "off". Then, abort these steps.
8. If `category` is Automatic, let the element's `autofill field name` be the string "on", let its `autofill hint set` be empty, and let its `IDL-exposed autofill value` be the string "on". Then, abort these steps.
9. Let `scope tokens` be an empty list.
10. Let `hint tokens` be an empty set.
11. Let `IDL value` have the same value as `field`.
12. If the `index`th token in `tokens` is the first entry, then skip to the step labeled *done*.
13. Decrement `index` by one.
14. If `category` is Contact and the `index`th token in `tokens` is an `ASCII case-insensitive` match for one of the strings in the following list, then run the substeps that follow:
 - "home"
 - "work"
 - "mobile"
 - "fax"
 - "pager"

The substeps are:

1. Let `contact` be the matching string from the list above.
 2. Insert `contact` at the start of `scope tokens`.
 3. Add `contact` to `hint tokens`.
 4. Let `IDL value` be the concatenation of `contact`, a U+0020 SPACE character, and the previous value of `IDL value` (which at this point will always be `field`).
 5. If the `index`th entry in `tokens` is the first entry, then skip to the step labeled *done*.
 6. Decrement `index` by one.
15. If the `index`th token in `tokens` is an `ASCII case-insensitive` match for one of the strings in the following list, then run the substeps that follow:
 - "shipping"
 - "billing"

The substeps are:

1. Let `mode` be the matching string from the list above.

2. Insert *mode* at the start of *scope tokens*.
 3. Add *mode* to *hint tokens*.
 4. Let *IDL value* be the concatenation of *mode*, a U+0020 SPACE character, and the previous value of *IDL value* (which at this point will either be *field* or the concatenation of *contact*, a space, and *field*).
 5. If the *index*th entry in *tokens* is the first entry, then skip to the step labeled *done*.
 6. Decrement *index* by one.
16. If the *index*th entry in *tokens* is not the first entry, then jump to the step labeled *default*.
 17. If the first eight characters of the *index*th token in *tokens* are not an [ASCII case-insensitive](#) match for the string "section-", then jump to the step labeled *default*.
 18. Let *section* be the *index*th token in *tokens*, [converted to ASCII lowercase](#).
 19. Insert *section* at the start of *scope tokens*.
 20. Let *IDL value* be the concatenation of *section*, a U+0020 SPACE character, and the previous value of *IDL value*.
 21. *Done*: Let the element's [autofill hint set](#) be *hint tokens*.
 22. Let the element's [autofill scope](#) be *scope tokens*.
 23. Let the element's [autofill field name](#) be *field*.
 24. Let the element's [IDL-exposed autofill value](#) be *IDL value*.
 25. Abort these steps.
 26. *Default*: Let the element's [IDL-exposed autofill value](#) be the empty string, and its [autofill hint set](#) and [autofill scope](#) be empty.
 27. If the element's [autocomplete](#) attribute is wearing the [autofill anchor mantle](#), then let the element's [autofill field name](#) be the empty string and abort these steps.
 28. Let *form* be the element's [form owner](#), if any, or null otherwise.
 29. If *form* is not null and *form*'s [autocomplete](#) attribute is in the [off](#) state, then let the element's [autofill field name](#) be "off".
- Otherwise, let the element's [autofill field name](#) be "on".



For the purposes of autofill, a **control's data** depends on the kind of control:

An [<input>](#) element with its [type](#) attribute in the [E-mail](#) state and with the [multiple](#) attribute specified

The element's [values](#).

Any other [<input>](#) element

A [<textarea>](#) element

The element's [value](#).

A [<select>](#) element with its [multiple](#) attribute specified

The [<option>](#) elements in the [<select>](#) element's [list of options](#) that have their [selectedness](#) set to true.

Any other [<select>](#) element

The [<option>](#) element in the [<select>](#) element's [list of options](#) that has its [selectedness](#) set to true.



How to process the [autofill hint set](#), [autofill scope](#), and [autofill field name](#) depends on the mantle that the [autocomplete](#) attribute is wearing.

↪ When wearing the [autofill expectation mantle](#)...

When an element's [autofill field name](#) is `"off"`, the user agent should not remember the [control's data](#), and should not offer past values to the user.

NOTE:

In addition, when an element's [autofill field name](#) is `"off"`, [values are reset](#) when [traversing the history](#).

EXAMPLE 530

Banks frequently do not want user agents to prefill login information:

```
<p><label>Account: <input type="text" name="ac" autocomplete="off"></label></p>
<p><label>PIN: <input type="password" name="pin" autocomplete="off"></label></p>
```

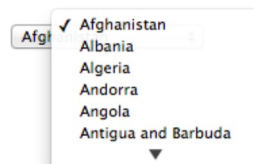
When an element's [autofill field name](#) is *not* `"off"`, the user agent may store the [control's data](#), and may offer previously stored values to the user.

EXAMPLE 531

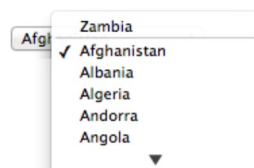
For example, suppose a user visits a page with this control:

```
<select name="country">
  <option>Afghanistan
  <option>Albania
  <option>Algeria
  <option>Andorra
  <option>Angola
  <option>Antigua and Barbuda
  <option>Argentina
  <option>Armenia
  <!-- ... -->
  <option>Yemen
  <option>Zambia
  <option>Zimbabwe
</select>
```

This might render as follows:



Suppose that on the first visit to this page, the user selects "Zambia". On the second visit, the user agent could duplicate the entry for Zambia at the top of the list, so that the interface instead looks like this:



When the [autofill field name](#) is "on", the user agent should attempt to use heuristics to determine the most appropriate values to offer the user, e.g., based on the element's [name](#) value, the position of the element in the document's DOM, what other fields exist in the form, and so forth.

When the [autofill field name](#) is one of the names of the [autofill fields](#) described above, the user agent should provide suggestions that match the meaning of the field name as given in the table earlier in this section. The [autofill hint set](#) should be used to select amongst multiple possible suggestions.

EXAMPLE 532

For example, if a user once entered one address into fields that used the "shipping" keyword, and another address into fields that used the "billing" keyword, then in subsequent forms only the first address would be suggested for form controls whose [autofill hint set](#) contains the keyword "shipping". Both addresses might be suggested, however, for address-related form controls whose [autofill hint set](#) does not contain either keyword.

↪ When wearing the [autofill anchor mantle](#)...

When the [autofill field name](#) is not the empty string, then the user agent must act as if the user had specified the [control's data](#) for the given [autofill hint set](#), [autofill scope](#), and [autofill field name](#) combination.

When the user agent **autofills form controls**, elements with the same [form owner](#) and the same [autofill scope](#) must use data relating to the same person, address, payment instrument, and contact details. When a user agent autofills "country" and "country-name" fields with the same [form owner](#) and [autofill scope](#), and the user agent has a value for the country" field(s), then the "country-name" field(s) must be filled using a human-readable name for the same country. When a user agent fills in multiple fields at once, all fields with the same [autofill field name](#), [form owner](#) and [autofill scope](#) must be filled with the same value.

EXAMPLE 533

Suppose a user agent knows of two phone numbers, +1 555 123 1234 and +1 555 666 7777. It would not be conforming for the user agent to fill a field with `autocomplete="shipping tel-local-prefix"` with the value "123" and another field in the same form with `autocomplete="shipping tel-local-suffix"` with the value "7777". The only valid pre-filled values given the aforementioned information would be "123" and "1234", or "666" and "7777", respectively.

EXAMPLE 534

Similarly, if a form for some reason contained both a "cc-exp" field and a "cc-exp-month" field, and the user agent pre-filled the form, then the month component of the former would have to match the latter.

EXAMPLE 535

This requirement interacts with the [autofill anchor mantle](#) also. Consider the following markup snippet:

```
<form>
  <input type=hidden autocomplete="nickname" value="TreePlate">
  <input type=text autocomplete="nickname">
</form>
```

The only value that a conforming user agent could suggest in the text field is "TreePlate", the value given by the hidden [<input>](#) element.

The "section-*" tokens in the [autofill scope](#) are opaque; user agents must not attempt to derive meaning from the precise values of these tokens.

EXAMPLE 536

For example, it would not be conforming if the user agent decided that it should offer the address it knows to be the user's daughter's address for "section-child" and the addresses it knows to be the user's spouses' addresses for "section-spouse".

The autocompletion mechanism must be implemented by the user agent acting as if the user had modified the [control's data](#), and must be done at a time where the element is *mutable* (e.g., just after the element has been [inserted into the document](#), or when the user agent [stops parsing](#)). User agents must only prefill controls using values that the user could have entered.

EXAMPLE 537

For example, if a `<select>` element only has `<option>` elements with values "Steve" and "Rebecca", "Jay", and "Bob", and has an [autofill field name](#) "given-name", but the user agent's only idea for what to prefill the field with is "Evan", then the user agent cannot prefill the field. It would not be conforming to somehow set the `<select>` element to the value "Evan", since the user could not have done so themselves.

A user agent prefilling a form control's [value](#) must not cause that control to [suffer from a type mismatch](#), [suffer from being too long](#), [suffer from being too short](#), [suffer from an underflow](#), [suffer from an overflow](#), [suffer from a step mismatch](#), or [suffer from a pattern mismatch](#). Where possible given the control's constraints, user agents must use the format given as canonical in the aforementioned table. Where it's not possible for the canonical format to be used, user agents should use heuristics to attempt to convert values so that they can be used.

EXAMPLE 538

For example, if the user agent knows that the user's middle name is "Ines", and attempts to prefill a form control that looks like this:

```
<input name=middle-initial maxlength=1 autocomplete="additional-name">
```

...then the user agent could convert "Ines" to "I" and prefill it that way.

EXAMPLE 539

A more elaborate example would be with month values. If the user agent knows that the user's birthday is the 27th of July 2012, then it might try to prefill all of the following controls with slightly different values, all driven from this information:

<code><input name=b type=month autocomplete="bday"></code>	2012-07	The day is dropped since the Month state only accepts a month/year combination.
<code><select name=c autocomplete="bday"> <option>Jan <option>Feb ... <option>Jul <option>Aug ... </select></code>	July	The user agent picks the month from the listed options, either by noticing there are twelve options and picking the 7th, or by recognizing that one of the strings (three characters "Jul" followed by a newline and a space) is a close match for the name of the month (July) in one of the user agent's supported languages, or through some other similar mechanism.
<code><input name=a type=number min=1 max=12 autocomplete="bday-month"></code>	7	User agent converts "July" to a month number in the range 1..12, like the field.
<code><input name=a type=number min=0 max=11 autocomplete="bday-month"></code>	6	User agent converts "July" to a month number in the range 0..11, like the field.
<code><input name=a type=number min=1 max=11 autocomplete="bday-month"></code>		User agent doesn't fill in the field, since it can't make a good guess as to what the form expects.

A user agent may allow the user to override an element's [autofill field name](#), e.g., to change it from "off" to "on" to allow values to be remembered and prefilled despite the page author's objections, or to always "off", never remembering values.

More specifically, user agents may in particular consider replacing the [autofill field name](#) of form controls that match the description given in the first column of the following table, when their [autofill field name](#) is either "on" or "off", with the value given in the second cell of that row. If this table is used, the replacements must be done in [tree order](#), since all but the first row references the [autofill field name](#) of earlier elements. When the descriptions below refer to form controls being preceded or followed by others, they mean in the list of [listed elements](#) that share the same [form owner](#).

Form control	New autofill field name
an <input> element whose type attribute is in the Text state that is followed by an <input> element whose type attribute is in the Password state	"username"
an <input> element whose type attribute is in the Password state that is preceded by an <input> element whose autofill field name is "username"	"current-password"
an <input> element whose type attribute is in the Password state that is preceded by an <input> element whose autofill field name is "current-password"	"new-password"
an <input> element whose type attribute is in the Password state that is preceded by an <input> element whose autofill field name is "new-password"	"new-password"

The **autocomplete** IDL attribute must [reflect](#) the content attribute of the same name.

§ 4.10.20. APIs for text field selections

The [<input>](#) and [<textarea>](#) elements define the following members in their DOM interfaces for handling their selection: [select\(\)](#), [selectionStart](#), [selectionEnd](#), [selectionDirection](#), [setRangeText\(replacement\)](#), [setSelectionRange\(start, end\)](#)

The [setRangeText\(\)](#) method uses the following enumeration:

```
enum SelectionMode {
  "select",
  "start",
  "end",
  "preserve" // default
};
```

These methods and attributes expose and control the selection of [<input>](#) and [<textarea>](#) text fields.

This definition is non-normative. Implementation requirements are given below this definition.

`element . select()`

Selects everything in the text field.

`element . selectionStart` [= `value`]

Returns the offset to the start of the selection.

Can be set, to change the start of the selection.

`element . selectionEnd` [= `value`]

Returns the offset to the end of the selection.

Can be set, to change the end of the selection.

`element . selectionDirection` [= `value`]

Returns the current direction of the selection.

Can be set, to change the direction of the selection.

The possible values are "forward", "backward", and "none".

`element.setSelectionRange(start, end [, direction ] )`

Changes the selection to cover the given substring in the given direction. If the direction is omitted, it will be reset to be the platform default (none or forward).

`element.setRangeText(replacement [, start, end [, selectionMode ] ] )`

Replaces a range of text with the new text. If the `start` and `end` arguments are not provided, the range is assumed to be the selection.

The final argument determines how the selection should be set after the text has been replaced. The possible values are:

"select"

Selects the newly inserted text.

"start"

Moves the selection to just before the inserted text.

"end"

Moves the selection to just after the selected text.

"preserve"

Attempts to preserve the selection. This is the default.

For `<input>` elements, calling these methods while they [don't apply](#), and getting or setting these attributes while they [don't apply](#), must throw an `InvalidStateError` exception. Otherwise, they must act as described below.

For `<input>` elements, these methods and attributes must operate on the element's [value](#). For `<textarea>` elements, these methods and attributes must operate on the element's [raw value](#).

Where possible, user interface features for changing the text selection in `<input>` and `<textarea>` elements must be implemented in terms of the DOM API described in this section, so that, e.g., all the same events fire.

The selections of `<input>` and `<textarea>` elements have a *direction*, which is either *forward*, *backward*, or *none*. This direction is set when the user manipulates the selection. The exact meaning of the selection direction depends on the platform.

NOTE:

On Windows, the direction indicates the position of the caret relative to the selection: a forward selection has the caret at the end of the selection and a backward selection has the caret at the start of the selection. Windows has no *none* direction. On Mac, the direction indicates which end of the selection is affected when the user adjusts the size of the selection using the arrow keys with the Shift modifier: the forward direction means the end of the selection is modified, and the backwards direction means the start of the selection is modified. The *none* direction is the default on Mac, it indicates that no particular direction has yet been selected. The user sets the direction implicitly when first adjusting the selection, based on which directional arrow key was used.

The `select()` method must cause the contents of the text field to be fully selected, with the selection direction being *none*, if the platform support selections with the direction *none*, or otherwise *forward*. The user agent must then [queue a task](#) to [fire a simple event](#) that bubbles named `select` at the element, using the [user interaction task source](#) as the task source.

In the case of `<input>` elements, if the control has no text field, then the method must do nothing.

EXAMPLE 540

For instance, in a user agent where `<input type=color>` is rendered as a color well with a picker, as opposed to a text field accepting a hexadecimal color code, there would be no text field, and thus nothing to select, and thus calls to the method are ignored.

The `selectionStart` attribute must, on getting, return the offset (in logical order) to the character that immediately follows the start of the selection. If there is no selection, then it must return the offset (in logical order) to the character that immediately follows the text entry cursor.

On setting, it must act as if the [setSelectionRange\(\)](#) method had been called, with the new value as the first argument; the current value of the [selectionEnd](#) attribute as the second argument, unless the current value of the [selectionEnd](#) is less than the new value, in which case the second argument must also be the new value; and the current value of the [selectionDirection](#) as the third argument.

The [selectionEnd](#) attribute must, on getting, return the offset (in logical order) to the character that immediately follows the end of the selection. If there is no selection, then it must return the offset (in logical order) to the character that immediately follows the text entry cursor.

On setting, it must act as if the [setSelectionRange\(\)](#) method had been called, with the current value of the [selectionStart](#) attribute as the first argument, the new value as the second argument, and the current value of the [selectionDirection](#) as the third argument.

The [selectionDirection](#) attribute must, on getting, return the string corresponding to the current selection direction: if the direction is *forward*, "forward"; if the direction is *backward*, "backward"; and otherwise, "none".

On setting, it must act as if the [setSelectionRange\(\)](#) method had been called, with the current value of the [selectionStart](#) IDL attribute as the first argument, the current value of the [selectionEnd](#) IDL attribute as the second argument, and the new value as the third argument.

The [setSelectionRange\(start , end , direction \)](#) method must set the selection of the text field to the sequence of characters starting with the character at the *start*th position (in logical order) and ending with the character at the (*end*-1)th position. Arguments greater than the length of the value of the text field must be treated as pointing at the end of the text field. If *end* is less than or equal to *start* then the start of the selection and the end of the selection must both be placed immediately before the character with offset *end*. In user agents where there is no concept of an empty selection, this must set the cursor to be just before the character with offset *end*. The direction of the selection must be set to *backward* if *direction* is a [case-sensitive](#) match for the string "backward", *forward* if *direction* is a [case-sensitive](#) match for the string "forward" or if the platform does not support selections with the direction *none*, and *none* otherwise (including if the argument is omitted). The user agent must then [queue a task](#) to [fire a simple event](#) that bubbles named `select` at the element, using the [user interaction task source](#) as the task source.

The [setRangeText\(replacement , start , end , selectMode \)](#) method must run the following steps:

1. If the method has only one argument, then let *start* and *end* have the values of the [selectionStart](#) IDL attribute and the [selectionEnd](#) IDL attribute respectively.
Otherwise, let *start*, *end* have the values of the second and third arguments respectively.
2. If *start* is greater than *end*, then throw an `IndexSizeError` exception and abort these steps.
3. If *start* is greater than the length of the value of the text field, then set it to the length of the value of the text field.
4. If *end* is greater than the length of the value of the text field, then set it to the length of the value of the text field.
5. Let *selection start* be the current value of the [selectionStart](#) IDL attribute.
6. Let *selection end* be the current value of the [selectionEnd](#) IDL attribute.
7. If *start* is less than *end*, delete the sequence of characters starting with the character at the *start*th position (in logical order) and ending with the character at the (*end*-1)th position.
8. Insert the value of the first argument into the text of the value of the text field, immediately before the *start*th character.
9. Let *new length* be the length of the value of the first argument.
10. Let *new end* be the sum of *start* and *new length*.
11. Run the appropriate set of substeps from the following list:

↪ If the fourth argument's value is "select"

Let *selection start* be *start*.

Let *selection end* be *new end*.

↪ If the fourth argument's value is **"start"**

Let *selection start* and *selection end* be *start*.

↪ If the fourth argument's value is **"end"**

Let *selection start* and *selection end* be *new end*.

↪ If the fourth argument's value is **"preserve"** (the default)

1. Let *old length* be *end* minus *start*.

2. Let *delta* be *new length* minus *old length*.

3. If *selection start* is greater than *end*, then increment it by *delta*. (If *delta* is negative, i.e., the new text is shorter than the old text, then this will *decrease* the value of *selection start*.)

Otherwise: if *selection start* is greater than *start*, then set it to *start*. (This snaps the start of the selection to the start of the new text if it was in the middle of the text that it replaced.)

4. If *selection end* is greater than *end*, then increment it by *delta* in the same way.

Otherwise: if *selection end* is greater than *start*, then set it to *new end*. (This snaps the end of the selection to the end of the new text if it was in the middle of the text that it replaced.)

12. Set the selection of the text field to the sequence of characters starting with the character at the *selection start*th position (in logical order) and ending with the character at the (*selection end*-1)th position. In user agents where there is no concept of an empty selection, this must set the cursor to be just before the character with offset *end*. The direction of the selection must be set to *forward* if the platform does not support selections with the direction *none*, and *none* otherwise.

13. [Queue a task to fire a simple event](#) that bubbles named `select` at the element, using the [user interaction task source](#) as the task source.

All elements to which this API [applies](#) have either a selection or a text entry cursor position at all times (even for elements that are not [being rendered](#)). User agents should follow platform conventions to determine their initial state.

Characters with no visible rendering, such as U+200D ZERO WIDTH JOINER, still count as characters. Thus, for instance, the selection can include just an invisible character, and the text insertion cursor can be placed to one side or another of such a character.

EXAMPLE 541

To obtain the currently selected text, the following JavaScript suffices:

```
var selectionText = control.value.substring(control.selectionStart, control.selectionEnd);
```

...where *control* is the `<input>` or `<textarea>` element.

EXAMPLE 542

To add some text at the start of a text control, while maintaining the text selection, the three attributes must be preserved:

```
var oldStart = control.selectionStart;
var oldEnd = control.selectionEnd;
var oldDirection = control.selectionDirection;
var prefix = "https://";
control.value = prefix + control.value;
control.setSelectionRange(oldStart + prefix.length, oldEnd + prefix.length, oldDirection);
```

...where *control* is the `<input>` or `<textarea>` element.

4.10.21.1. Definitions

A [submittable element](#) is a **candidate for constraint validation** except when a condition has **barred the element from constraint validation**. (For example, an element is [barred from constraint validation](#) if it is an `<object>` element.)

An element can have a **custom validity error message** defined. Initially, an element must have its [custom validity error message](#) set to the empty string. When its value is not the empty string, the element is [suffering from a custom error](#). It can be set using the [setCustomValidity\(\)](#) method. The user agent should use the [custom validity error message](#) when alerting the user to the problem with the control.

An element can be constrained in various ways. The following is the list of **validity states** that a form control can be in, making the control invalid for the purposes of constraint validation. (The definitions below are non-normative; other parts of this specification define more precisely when each state applies or does not.)

Suffering from being missing

When a control has no [value](#) but has a required attribute (`<input>` [required](#), `<textarea>` [required](#)); or, in the case of an element in a *radio button group*, any of the other elements in the group has a required attribute; or, for `<select>` elements, none of the `<option>` elements have their [selectedness](#) set (`<select>` [required](#)).

Suffering from a type mismatch

When a control that allows arbitrary user input has a [value](#) that is not in the correct syntax ([E-mail](#), [URL](#)).

Suffering from a pattern mismatch

When a control has a [value](#) that doesn't satisfy the [pattern](#) attribute.

Suffering from being too long

When a control has a [value](#) that is too long for the [form control maxlength attribute](#) (`<input>` [maxlength](#), `<textarea>` [maxlength](#)).

Suffering from being too short

When a control has a [value](#) that is too short for the [form control minlength attribute](#) (`<input>` [minlength](#), `<textarea>` [minlength](#)).

Suffering from an underflow

When a control has a [value](#) that is not the empty string and is too low for the [min](#) attribute.

Suffering from an overflow

When a control has a [value](#) that is not the empty string and is too high for the [max](#) attribute.

Suffering from a step mismatch

When a control has a [value](#) that doesn't fit the rules given by the [step](#) attribute.

Suffering from bad input

When a control has incomplete input and the user agent does not think the user ought to be able to submit the form in its current state.

Suffering from a custom error

When a control's [custom validity error message](#) (as set by the element's [setCustomValidity\(\)](#) method) is not the empty string.

NOTE:

An element can still suffer from these states even when the element is disabled; thus these states can be represented in the DOM even if validating the form during submission wouldn't indicate a problem to the user.

An element **satisfies its constraints** if it is not suffering from any of the above [validity states](#).

4.10.21.2. Constraint validation

When the user agent is required to **statically validate the constraints** of `<form>` element *form*, it must run the following steps, which return either a *positive* result (all the controls in the form are valid) or a *negative* result (there are invalid controls) along with a (possibly empty) list of elements that are invalid and for which no script has claimed responsibility:

1. Let *controls* be a list of all the [submittable elements](#) whose [form owner](#) is *form*, in [tree order](#).
2. Let *invalid controls* be an initially empty list of elements.
3. For each element *field* in *controls*, in [tree order](#), run the following substeps:
 1. If *field* is not a [candidate for constraint validation](#), then move on to the next element.
 2. Otherwise, if *field* [satisfies its constraints](#), then move on to the next element.
 3. Otherwise, add *field* to *invalid controls*.
4. If *invalid controls* is empty, then return a *positive* result and abort these steps.
5. Let *unhandled invalid controls* be an initially empty list of elements.
6. For each element *field* in *invalid controls*, if any, in [tree order](#), run the following substeps:
 1. [Fire a simple event](#) named `invalid` that is cancelable at *field*.
 2. If the event was not canceled, then add *field* to *unhandled invalid controls*.
7. Return a *negative* result with the list of elements in the *unhandled invalid controls* list.

If a user agent is to **interactively validate the constraints** of `<form>` element *form*, then the user agent must run the following steps:

1. [Statically validate the constraints](#) of *form*, and let *unhandled invalid controls* be the list of elements returned if the result was *negative*.
2. If the result was *positive*, then return that result and abort these steps.
3. Report the problems with the constraints of at least one of the elements given in *unhandled invalid controls* to the user. User agents may focus one of those elements in the process, by running the [focusing steps](#) for that element, and may change the scrolling position of the document, or perform some other action that brings the element to the user's attention. User agents may report more than one constraint violation. User agents may coalesce related constraint violation reports if appropriate (e.g., if multiple radio buttons in a [group](#) are marked as required, only one error need be reported). If one of the controls is not [being rendered](#) (e.g., it has the `hidden` attribute set) then user agents may report a script error.
4. Return a *negative* result.

4.10.21.3. The constraint validation API

This definition is non-normative. Implementation requirements are given below this definition.

`element`.willValidate

Returns true if the element will be validated when the form is submitted; false otherwise.

`element`.{{HTMLElement/setCustomValidity(*message*)}}

Sets a custom error, so that the element would fail to validate. The given message is the message to be shown to the user when reporting the problem to the user.

If the argument is the empty string, clears the custom error.

`element`.validity.valueMissing

Returns true if the element has no value but is a required field; false otherwise.

`element`.validity.typeMismatch

Returns true if the element's value is not in the correct syntax; false otherwise.

`element`.validity.patternMismatch

Returns true if the element's value doesn't match the provided pattern; false otherwise.

`element`.validity.tooLong

Returns true if the element's value is longer than the provided maximum length; false otherwise.

`element . validity . tooShort`

Returns true if the element's value, if it is not the empty string, is shorter than the provided minimum length; false otherwise.

`element . validity . rangeUnderflow`

Returns true if the element's value is lower than the provided minimum; false otherwise.

`element . validity . rangeOverflow`

Returns true if the element's value is higher than the provided maximum; false otherwise.

`element . validity . stepMismatch`

Returns true if the element's value doesn't fit the rules given by the [step](#) attribute; false otherwise.

`element . validity . badInput`

Returns true if the user has provided input in the user interface that the user agent is unable to convert to a value; false otherwise.

`element . validity . customError`

Returns true if the element has a custom error; false otherwise.

`element . validity . valid`

Returns true if the element's value has no validity problems; false otherwise.

`valid = element . checkValidity()`

Returns true if the element's value has no validity problems; false otherwise. Fires an `invalid` event at the element in the latter case.

`valid = element . reportValidity()`

Returns true if the element's value has no validity problems; otherwise, returns false, fires an `invalid` event at the element, and (if the event isn't canceled) reports the problem to the user.

`element . validationMessage`

Returns the error message that would be shown to the user if the element was to be checked for validity.

The **`willValidate`** IDL attribute must return true if an element is a [candidate for constraint validation](#), and false otherwise (i.e., false if any conditions are [barring it from constraint validation](#)).

The **`setCustomValidity(message)`**, when invoked, must set the [custom validity error message](#) to the value of the given *message* argument.

EXAMPLE 543

In the following example, a script checks the value of a form control each time it is edited, and whenever it is not a valid value, uses the [setCustomValidity\(\)](#) method to set an appropriate message.

```
<label>Feeling: <input name=f type="text" oninput="check(this)"></label>
<script>
  function check(input) {
    if (input.value == "good" ||
        input.value == "fine" ||
        input.value == "tired") {
      input.setCustomValidity('' + input.value + ' is not a feeling.');
```

```
    } else {
      // input is fine -- reset the error message
      input.setCustomValidity('');
    }
  }
}
```

```
</script>
```

The **validity** IDL attribute must return a [ValidityState](#) object that represents the [validity states](#) of the element. This object is [live](#).

```
interface ValidityState {
  readonly attribute boolean valueMissing;
  readonly attribute boolean typeMismatch;
  readonly attribute boolean patternMismatch;
  readonly attribute boolean tooLong;
  readonly attribute boolean tooShort;
  readonly attribute boolean rangeUnderflow;
  readonly attribute boolean rangeOverflow;
  readonly attribute boolean stepMismatch;
  readonly attribute boolean badInput;
  readonly attribute boolean customError;
  readonly attribute boolean valid;
};
```

A [ValidityState](#) object has the following attributes. On getting, they must return true if the corresponding condition given in the following list is true, and false otherwise.

valueMissing, of type [boolean](#), [readonly](#)

The control is [suffering from being missing](#).

typeMismatch, of type [boolean](#), [readonly](#)

The control is [suffering from a type mismatch](#).

patternMismatch, of type [boolean](#), [readonly](#)

The control is [suffering from a pattern mismatch](#).

tooLong, of type [boolean](#), [readonly](#)

The control is [suffering from being too long](#).

tooShort, of type [boolean](#), [readonly](#)

The control is [suffering from being too short](#).

rangeUnderflow, of type [boolean](#), [readonly](#)

The control is [suffering from an underflow](#).

rangeOverflow, of type [boolean](#), [readonly](#)

The control is [suffering from an overflow](#).

stepMismatch, of type [boolean](#), [readonly](#)

The control is [suffering from a step mismatch](#).

badInput, of type [boolean](#), [readonly](#)

The control is [suffering from bad input](#).

customError, of type [boolean](#), [readonly](#)

The control is [suffering from a custom error](#).

valid, of type [boolean](#), [readonly](#)

None of the other conditions are true.

When the **checkValidity()** method is invoked, if the element is a [candidate for constraint validation](#) and does not [satisfy its constraints](#), the user agent must [fire a simple event](#) named `invalid` that is cancelable (but in this case has no default action) at the element and return false. Otherwise, it must only return true without doing anything else.

When the **reportValidity()** method is invoked, if the element is a [candidate for constraint validation](#) and does not [satisfy its constraints](#), the user agent must: [fire a simple event](#) named `invalid` that is cancelable at the element, and if that event is not canceled, report the problems with the constraints of that element to the user; then, return false. Otherwise, it must only return true without doing anything else. When reporting the problem with the constraints to the user, the user agent may run the [focusing steps](#) for that element, and may change the scrolling position of the document, or perform some other action that brings the element to the user's attention. User agents may report more than one constraint violation, if the element suf-

fers from multiple problems at once. If the element is not [being rendered](#), then the user agent may, instead of notifying the user, report a script error.

The **validationMessage** attribute must return the empty string if the element is not a [candidate for constraint validation](#) or if it is one but it [satisfies its constraints](#); otherwise, it must return a suitably localized message that the user agent would show the user if this were the only form control with a validity constraint problem. If the user agent would not actually show a textual message in such a situation (e.g., it would show a graphical cue instead), then the attribute must return a suitably localized message that expresses (one or more of) the validity constraint(s) that the control does not satisfy. If the element is a [candidate for constraint validation](#) and is [suffering from a custom error](#), then the [custom validity error message](#) should be present in the return value.

§ 4.10.21.4. Security

Servers should not rely on client-side validation. Client-side validation can be intentionally bypassed by hostile users, and unintentionally bypassed by users of older user agents or automated tools that do not implement these features. The constraint validation features are only intended to improve the user experience, not to provide any kind of security mechanism.

§ 4.10.22. Form submission

§ 4.10.22.1. Introduction

This section is non-normative.

When a form is submitted, the data in the form is converted into the structure specified by the [enctype](#), and then sent to the destination specified by the [action](#) using the given [method](#).

For example, take the following form:

```
<form action="/find.cgi" method=get>
  <input type=text name=t>
  <input type=search name=q>
  <input type=submit>
</form>
```

If the user types in "cats" in the first field and "fur" in the second, and then hits the submit button, then the user agent will load `/find.cgi?t=cats&q=fur`.

On the other hand, consider this form:

```
<form action="/find.cgi" method=post enctype="multipart/form-data">
  <input type=text name=t>
  <input type=search name=q>
  <input type=submit>
</form>
```

Given the same user input, the result on submission is quite different: the user agent instead does an HTTP POST to the given URL, with as the entity body something like the following text:

```
-----kYFr4jNJEgCervEContent-Disposition: form-data; name="t"

cats
-----kYFr4jNJEgCervE
Content-Disposition: form-data; name="q"

fur
-----kYFr4jNJEgCervE--
```

§ 4.10.22.2. Implicit submission

A `<form>` element's **default button** is the first [submit button](#) in [tree order](#) whose [form owner](#) is that `<form>` element.

If the user agent supports letting the user submit a form implicitly (for example, on some platforms hitting the "enter" key while a text field is [focused](#) implicitly submits the form), then doing so for a form whose [default button](#) has a defined [activation behavior](#) must cause the user agent to [run synthetic click activation steps](#) on that [default button](#).

NOTE:

Consequently, if the [default button](#) is disabled, the form is not submitted when such an implicit submission mechanism is used. (A button has no [activation behavior](#) when disabled.)

NOTE:

There are pages on the Web that are only usable if there is a way to implicitly submit forms, so user agents are strongly encouraged to support this.

If the form has no [submit button](#), then the implicit submission mechanism must do nothing if the form has more than one *field that blocks implicit submission*, and must [submit](#) the `<form>` element from the `<form>` element itself otherwise.

For the purpose of the previous paragraph, an element is a *field that blocks implicit submission* of a `<form>` element if it is an `<input>` element whose [form owner](#) is that `<form>` element and whose [type](#) attribute is in one of the following states: [Text](#), [Search](#), [URL](#), [Telephone](#), [E-mail](#), [Password](#), [Date](#), [Month](#), [Week](#), [Time](#), [Number](#)

§ 4.10.22.3. Form submission algorithm

When a `<form>` element *form* is **submitted** from an element *submitter* (typically a button), optionally with a *submitted* from [submit\(\)](#) method flag set, the user agent must run the following steps:

1. Let *form document* be the *form*'s [node document](#).
2. If *form document* has no associated [browsing context](#) or its [active sandboxing flag set](#) has its [sandboxed forms browsing context flag](#) set, then abort these steps without doing anything.
3. Let *form browsing context* be the [browsing context](#) of *form document*.
4. If the *submitted from submit() method* flag is not set, and the *submitter* element's [no-validate state](#) is false, then [interactively validate the constraints](#) of *form* and examine the result: if the result is negative (the constraint validation concluded that there were invalid fields and probably informed the user of this) then [fire a simple event](#) named *invalid* at the *form* element and then abort these steps.
5. If the *submitted from submit() method* flag is not set, then [fire a simple event](#) that bubbles and is cancelable named *submit*, at *form*. If the event's default action is prevented (i.e., if the event is canceled) then abort these steps. Otherwise, continue (effectively the default action is to perform the submission).
6. Let *form data set* be the result of [constructing the form data set](#) for *form* in the context of *submitter*.
7. Let *action* be the *submitter* element's [action](#).
8. If *action* is the empty string, let *action* be the [document's address](#) of the *form document*.
9. [Parse](#) the [URL](#) *action*, relative to the *submitter* element's [node document](#). If this fails, abort these steps.
10. Let *action* be the [resulting URL string](#).
11. Let *action components* be the [resulting URL record](#).
12. Let *scheme* be the [scheme](#) of the [resulting URL record](#).
13. Let *enctype* be the *submitter* element's [enctype](#).
14. Let *method* be the *submitter* element's [method](#).
15. Let *target* be the *submitter* element's [target](#).
16. If the user indicated a specific [browsing context](#) to use when submitting the form, then let *target browsing context* be

that [browsing context](#). Otherwise, apply [the rules for choosing a browsing context given a browsing context name](#) using *target* as the name and *form browsing context* as the context in which the algorithm is executed, and let *target browsing context* be the resulting [browsing context](#).

17. If *target browsing context* was created in the previous step, or, alternatively, if the *form document* has not yet [completely loaded](#) and the *submitted from* [submit\(\)](#) *method* flag is set, then let *replace* be true. Otherwise, let it be false.
18. Otherwise, select the appropriate row in the table below based on the value of *scheme* as given by the first cell of each row. Then, select the appropriate cell on that row based on the value of *method* as given in the first cell of each column. Then, jump to the steps named in that cell and defined below the table.

	GET	POST
http	Mutate action URL	Submit as entity body
https	Mutate action URL	Submit as entity body
ftp	Get action URL	Get action URL
javascript	Get action URL	Get action URL
data	Get action URL	Post to data:
mailto	Mail with headers	Mail as body

If *scheme* is not one of those listed in this table, then the behavior is not defined by this specification. User agents should, in the absence of another specification defining this, act in a manner analogous to that defined in this specification for similar schemes.

Each `<form>` element has a **planned navigation**, which is either null or a [task](#); when the `<form>` is first created, its [planned navigation](#) must be set to null. In the behaviors described below, when the user agent is required to **plan to navigate** to a particular resource *destination*, it must run the following steps:

1. If the `<form>` has a non-null [planned navigation](#), remove it from its [task queue](#).
2. Let the `<form>`'s [planned navigation](#) be a new [task](#) that consists of running the following steps:
 1. Let the `<form>`'s [planned navigation](#) be null.
 2. [Navigate](#) *target browsing context* to *destination*. If *replace* is true, then *target browsing context* must be navigated with [replacement enabled](#).

For the purposes of this task, *target browsing context* and *replace* are the variables that were set up when the overall [form submission algorithm](#) was run, with their values as they stood when this [planned navigation](#) was [queued](#).

3. [Queue a task](#) that is the `<form>`'s new [planned navigation](#).

The [task source](#) for this task is the [DOM manipulation task source](#).

The behaviors are as follows:

Mutate action URL

Let *query* be the result of encoding the *form data set* using the [application/x-www-form-urlencoded encoding algorithm](#), interpreted as a US-ASCII string.

Set *parsed action*'s [query](#) component to *query*.

Let *destination* be a new [URL](#) formed by applying the [URL serializer](#) algorithm to *parsed action*.

[Plan to navigate](#) to *destination*.

Submit as entity body

Let *entity body* be the result of encoding the *form data set* using the [appropriate form encoding algorithm](#).

Let *MIME type* be determined as follows:

If *enctype* is `application/x-www-form-urlencoded`

Let *MIME type* be "application/x-www-form-urlencoded".

If *enctype* is `multipart/form-data`

Let *MIME type* be the concatenation of the string "multipart/form-data;", a U+0020 SPACE character, the string "boundary=", and the [multipart/form-data boundary string](#) generated by the [multipart/form-data encoding algorithm](#).

If *enctype* is `text/plain`

Let *MIME type* be "text/plain".

Otherwise, [plan to navigate](#) to a new [request](#) whose *URL* is *action*, *method* is *method*, *header list* consists of Content-Type/*MIME type*, and *body* is *entity body*.

Get action URL

[Plan to navigate](#) to *action*.

NOTE:

The *form data set* is discarded.

Post to data:

Let *data* be the result of encoding the *form data set* using the [appropriate form encoding algorithm](#).

If *action* contains the string "%%%" (four U+0025 PERCENT SIGN characters), then [percent encode](#) all bytes in *data* that, if interpreted as US-ASCII, are not characters in the URL [default encode set](#), and then, treating the result as a US-ASCII string, [UTF-8 percent encode](#) all the U+0025 PERCENT SIGN characters in the resulting string and replace the first occurrence of "%%%" in *action* with the resulting doubly-escaped string. [\[URL\]](#)

Otherwise, if *action* contains the string "%" (two U+0025 PERCENT SIGN characters in a row, but not four), then [UTF-8 percent encode](#) all characters in *data* that, if interpreted as US-ASCII, are not characters in the URL [default encode set](#), and then, treating the result as a US-ASCII string, replace the first occurrence of "%" in *action* with the resulting escaped string. [\[URL\]](#)

[Plan to navigate](#) to the potentially modified *action* (which will be a `data: URL`).

Mail with headers

Let *headers* be the resulting encoding the *form data set* using the [application/x-www-form-urlencoded encoding algorithm](#), interpreted as a US-ASCII string.

Replace occurrences of U+002B PLUS SIGN characters (+) in *headers* with the string "%20".

Let *destination* consist of all the characters from the first character in *action* to the character immediately before the first U+003F QUESTION MARK character (?), if any, or the end of the string if there are none.

Append a single U+003F QUESTION MARK character (?) to *destination*.

Append *headers* to *destination*.

[Plan to navigate](#) to *destination*.

Mail as body

Let *body* be the resulting of encoding the *form data set* using the [appropriate form encoding algorithm](#) and then [percent encoding](#) all the bytes in the resulting byte string that, when interpreted as US-ASCII, are not characters in the URL [default encode set](#). [\[URL\]](#)

Let *destination* have the same value as *action*.

If *destination* does not contain a U+003F QUESTION MARK character (?), append a single U+003F QUESTION MARK character (?) to *destination*. Otherwise, append a single U+0026 AMPERSAND character (&).

Append the string "body=" to *destination*.

Append *body*, interpreted as a US-ASCII string, to *destination*.

[Plan to navigate](#) to *destination*.

The **appropriate form encoding algorithm** is determined as follows:

If *enctype* is `application/x-www-form-urlencoded`

Use the [application/x-www-form-urlencoded encoding algorithm](#).

If *enctype* is `multipart/form-data`

Use the [multipart/form-data encoding algorithm](#).

If *enctype* is `text/plain`

Use the [text/plain encoding algorithm](#).

§ 4.10.22.4. Constructing the form data set

The algorithm to **construct the form data set** for a form *form* optionally in the context of a submitter *submitter* is as follows. If not specified otherwise, *submitter* is null.

1. Let *controls* be a list of all the [submittable elements](#) whose [form owner](#) is *form*, in [tree order](#).
2. Let the *form data set* be a list of name-value-type tuples, initially empty.
3. *Loop*: For each element *field* in *controls*, in [tree order](#), run the following substeps:
 1. If any of the following conditions are met, then skip these substeps for this element:
 - The *field* element has a [<datalist>](#) element ancestor.
 - The *field* element is disabled.
 - The *field* element is a [button](#) but it is not *submitter*.
 - The *field* element is an [<input>](#) element whose [type](#) attribute is in the [Checkbox](#) state and whose [checkedness](#) is false.
 - The *field* element is an [<input>](#) element whose [type](#) attribute is in the [Radio Button](#) state and whose [checkedness](#) is false.
 - The *field* element is not an [<input>](#) element whose [type](#) attribute is in the [image button](#) state, and either the *field* element does not have a [name](#) attribute specified, or its [name](#) attribute's value is the empty string.
 - The *field* element is an [<object>](#) element that is not using a [plugin](#).

Otherwise, process *field* as follows:

2. Let *type* be the value of the [type](#) IDL attribute of *field*.
3. If the *field* element is an [<input>](#) element whose [type](#) attribute is in the [image button](#) state, then run these further nested substeps:
 1. If the *field* element has a [name](#) attribute specified and its value is not the empty string, let *name* be that value followed by a single U+002E FULL STOP character (.). Otherwise, let *name* be the empty string.
 2. Let *name_x* be the string consisting of the concatenation of *name* and a single U+0078 LATIN SMALL LETTER X character (x).
 3. Let *name_y* be the string consisting of the concatenation of *name* and a single U+0079 LATIN SMALL LETTER Y character (y).
 4. The *field* element is *submitter*, and before this algorithm was invoked the user [indicated a coordinate](#). Let *x* be the *x*-component of the coordinate selected by the user, and let *y* be the *y*-component of the coordinate selected by the user.
 5. Append an entry to the *form data set* with the name *name_x*, the value *x*, and the type *type*.
 6. Append an entry to the *form data set* with the name *name_y* and the value *y*, and the type *type*.

7. Skip the remaining substeps for this element: if there are any more elements in *controls*, return to the top of the *loop* step, otherwise, jump to the *end* step below.
4. Let *name* be the value of the *field* element's *name* attribute.
5. If the *field* element is a *<select>* element, then for each *<option>* element in the *<select>* element's *list of options* whose *selectedness* is true and that is not disabled, append an entry to the *form data set* with the *name* as the name, the *value* of the *<option>* element as the value, and *type* as the type.
6. Otherwise, if the *field* element is an *<input>* element whose *type* attribute is in the *Checkbox* state or the *Radio Button* state, then run these further nested substeps:
 1. If the *field* element has a *value* attribute specified, then let *value* be the value of that attribute; otherwise, let *value* be the string "on".
 2. Append an entry to the *form data set* with *name* as the name, *value* as the value, and *type* as the type.
7. Otherwise, if the *field* element is an *<input>* element whose *type* attribute is in the *File Upload* state, then for each file *selected* in the *<input>* element, append an entry to the *form data set* with the *name* as the name, the file (consisting of the name, the type, and the body) as the value, and *type* as the type. If there are no *selected files*, then append an entry to the *form data set* with the *name* as the name, the empty string as the value, and *application/octet-stream* as the type.
8. Otherwise, if the *field* element is an *<object>* element: try to obtain a *form submission* value from the *plugin*, and if that is successful, append an entry to the *form data set* with *name* as the name, the returned *form submission* value as the value, and the string "object" as the type.
9. Otherwise, append an entry to the *form data set* with *name* as the name, the *value* of the *field* element as the value, and *type* as the type.
10. If the element has a *dirname* attribute, and that attribute's value is not the empty string, then run these substeps:
 1. Let *dirname* be the value of the element's *dirname* attribute.
 2. Let *dir* be the string "ltr" if the *directionality* of the element is "ltr", and "rtl" otherwise (i.e., when the *directionality* of the element is "rtl").
 3. Append an entry to the *form data set* with *dirname* as the name, *dir* as the value, and the string "direction" as the type.

NOTE:

An element can only have a *dirname* attribute if it is a *<textarea>* element or an *<input>* element whose *type* attribute is in either the *Text* state or the *Search* state.

4. *End*: For the name of each entry in the *form data set*, and for the value of each entry in the *form data set* whose type is not "file" or "textarea", replace every occurrence of a U+000D CARRIAGE RETURN (CR) character not followed by a U+000A LINE FEED (LF) character, and every occurrence of a U+000A LINE FEED (LF) character not preceded by a U+000D CARRIAGE RETURN (CR) character, by a two-character string consisting of a U+000D CARRIAGE RETURN U+000A LINE FEED (CRLF) character pair.

NOTE:

In the case of the *value* of *<textarea>* elements, this newline normalization is already performed during the conversion of the control's *raw value* into the control's *value* (which also performs any necessary line wrapping). In the case of *<input>* elements *type* attributes in the *File Upload* state, the value is not normalized.

5. Return the *form data set*.

4.10.22.5. Selecting a *form submission* encoding

If the user agent is to **pick an encoding for a form**, it must run the following steps:

1. Let *encoding* be the *document's character encoding*.

2. If the form element has an `accept-charset` attribute, set `encoding` to the return value of running these substeps:
 1. Let `input` be the value of the form element's `accept-charset` attribute.
 2. Let `candidate encoding labels` be the result of [splitting `input` on spaces](#).
 3. Let `candidate encodings` be an empty list of [character encodings](#).
 4. For each token in `candidate encoding labels` in turn (in the order in which they were found in `input`), [get an encoding](#) for the token and, if this does not result in failure, append the [encoding](#) to `candidate encodings`.
 5. If `candidate encodings` is empty, return [UTF-8](#).
 6. Return the first encoding in `candidate encodings`.
3. Return the result of [getting an output encoding](#) from `encoding`.

§ 4.10.22.6. URL-encoded form data

See the WHATWG URL standard for details on `application/x-www-form-urlencoded`. [\[URL\]](#)

The **`application/x-www-form-urlencoded` encoding algorithm** is as follows:

1. Let `encoding` be the result of [picking an encoding for the form](#).
2. Let `serialized` be the result of running the [application/x-www-form-urlencoded serializer](#) given `form data set` and `encoding`.
3. Return the result of [encoding](#) `serialized`.

§ 4.10.22.7. Multipart form data

The **`multipart/form-data` encoding algorithm** is as follows:

1. Let `result` be the empty string.
2. If the algorithm was invoked with an explicit character encoding, let the selected character encoding be that encoding. (This algorithm is used by other specifications, which provide an explicit character encoding to avoid the dependency on the `<form>` element described in the next paragraph.)

Otherwise, if the `<form>` element has an `accept-charset` attribute, let the selected character encoding be the result of [picking an encoding for the form](#).

Otherwise, if the `<form>` element has no `accept-charset` attribute, but the [document's character encoding](#) is an [ASCII-compatible encoding](#), then that is the selected character encoding.

Otherwise, let the selected character encoding be UTF-8.

3. Let `charset` be the name of the selected [character encoding](#).
4. For each entry in the `form data set`, perform these substeps:
 1. If the entry's name is `"_charset_"` and its type is `"hidden"`, replace its value with `charset`.
 2. For each character in the entry's name and value that cannot be expressed using the selected character encoding, replace the character by a string consisting of a U+0026 AMPERSAND character (`&`), a U+0023 NUMBER SIGN character (`#`), one or more [ASCII digits](#) representing the Unicode code point of the character in base ten, and finally a U+003B SEMICOLON character (`;`).
5. Encode the (now mutated) `form data set` using the rules described by RFC 7578, *Returning Values from Forms: multipart/form-data*, and return the resulting byte stream. [\[RFC7578\]](#)

Each entry in the `form data set` is a *field*, the name of the entry is the *field name* and the value of the entry is the *field value*.

The order of parts must be the same as the order of fields in the *form data set*. Multiple entries with the same name must be treated as distinct fields.

The parts of the generated *multipart/form-data* resource that correspond to non-file fields must not have a *Content-Type* header specified. Their names and values must be encoded using the character encoding selected above.

File names included in the generated *multipart/form-data* resource (as part of file fields) must use the character encoding selected above, though the precise name may be approximated if necessary (e.g., newlines could be removed from file names, quotes could be changed to "%22", and characters not expressible in the selected character encoding could be replaced by other characters).

The boundary used by the user agent in generating the return value of this algorithm is the **multipart/form-data boundary string**. (This value is used to generate the MIME type of the [form submission](#) payload generated by this algorithm.)

For details on how to interpret *multipart/form-data* payloads, see RFC 7578. [\[RFC7578\]](#)

§ 4.10.22.8. Plain text form data

The **text/plain encoding algorithm** is as follows:

1. Let *result* be the empty string.
2. Let *encoding* be the result of [picking an encoding for the form](#).
3. Let *charset* be the name of *encoding*.
4. If the entry's name is "_charset_" and its type is "hidden", replace its value with *charset*.
5. If the entry's type is "file", replace its value with the file's name only.
6. For each entry in the *form data set*, perform these substeps:
 1. Append the entry's name to *result*.
 2. Append a single U+003D EQUALS SIGN character (=) to *result*.
 3. Append the entry's value to *result*.
 4. Append a U+000D CARRIAGE RETURN (CR) U+000A LINE FEED (LF) character pair to *result*.
7. Return the result of [encoding result](#) using *encoding*.

Payloads using the *text/plain* format are intended to be human readable. They are not reliably interpretable by computer, as the format is ambiguous (for example, there is no way to distinguish a literal newline in a value from the newline at the end of the value).

§ 4.10.23. Resetting a form

When a `<form>` element *form* is **reset**, the user agent must [fire a simple event](#) named *reset*, that bubbles and is cancelable, at *form*, and then, if that event is not canceled, must invoke the [reset algorithm](#) of each [resettable element](#) whose [form owner](#) is *form*.

When the [reset](#) algorithm is invoked by the `reset()` method, the *reset* event fired by the [reset](#) algorithm must not be [trusted](#).

Each [resettable element](#) defines its own **reset algorithm**. Changes made to form controls as part of these algorithms do not count as changes caused by the user (and thus, e.g., do not cause input events to fire).

§ 4.11. Interactive elements

§ 4.11.1. The details element

Categories:

[Flow content.](#)
[Sectioning root.](#)
[Interactive content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [flow content](#) is expected.

Content model:

One [<summary>](#) element followed by [flow content](#).

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)
 open - Whether the details are visible

Allowed ARIA role attribute values:

[Any role](#) that supports aria-expanded.

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)
 Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLDetailsElement : HTMLElement {
  attribute boolean open;
};
```

The [<details>](#) element [represents](#) a disclosure widget from which the user can obtain additional information or controls.

NOTE:

The [<details>](#) element is not appropriate for footnotes. Please see [§4.13.5 Footnotes](#) for details on how to mark up footnotes.

The first [<summary>](#) element child of the element, if any, [represents](#) the summary or legend of the details. If there is no child [<summary>](#) element, the user agent should provide its own legend (e.g., "Details").

The rest of the element's contents [represents](#) the additional information or controls.

The **open** content attribute is a [boolean attribute](#). If present, it indicates that both the summary and the additional information is to be shown to the user. If the attribute is absent, only the summary is to be shown.

When the element is created, if the attribute is absent, the additional information should be hidden; if the attribute is present, that information should be shown. Subsequently, if the attribute is removed, then the information should be hidden; if the attribute is added, the information should be shown.

The user agent should allow the user to request that the additional information be shown or hidden. To honor a request for the details to be shown, the user agent must set the open attribute on the element to the value open. To honor a request for the information to be hidden, the user agent must remove the open attribute from the element.

Whenever the open attribute is added to or removed from a [<details>](#) element, the user agent must [queue a task](#) that runs the following steps, which are known as the **details notification task steps**, for this [<details>](#) element:

1. If another [task](#) has been [queued](#) to run the [details notification task steps](#) for this [<details>](#) element, then abort these steps.

NOTE:

When the open attribute is toggled several times in succession, these steps essentially get coalesced so that only one event is fired.

2. [Fire a simple event](#) named toggle at the `<details>` element.

The [task source](#) for this task must be the [DOM manipulation task source](#).

The **open** IDL attribute must [reflect](#) the open content attribute.

EXAMPLE 544

The following example shows the `<details>` element being used to hide technical details in a progress report.

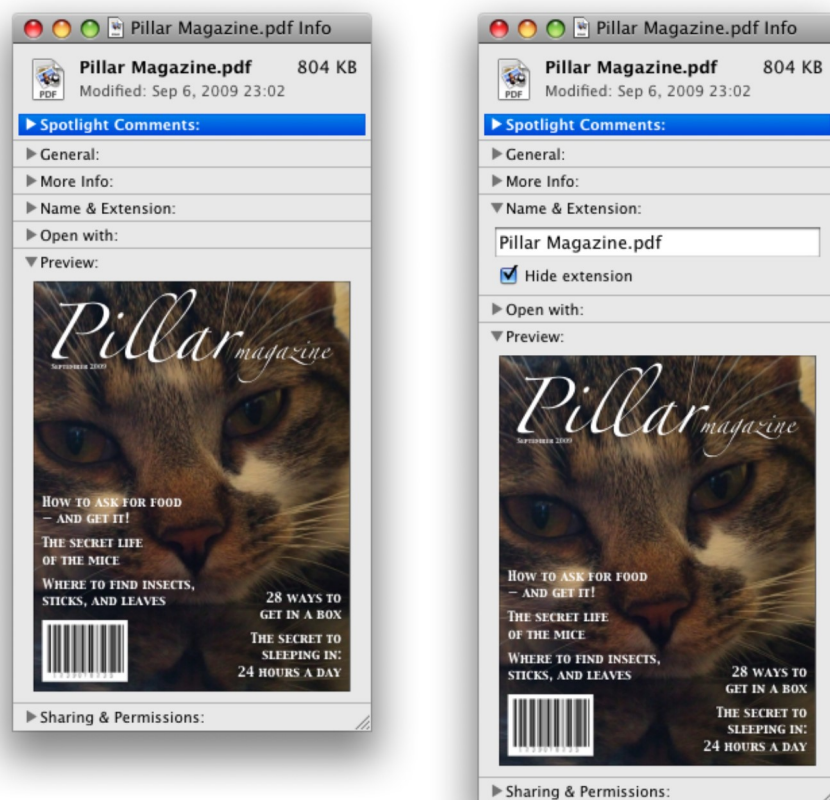
```
<section class="progress window">
  <h1>Copying "Really Achieving Your Childhood Dreams"</h1>
  <details>
    <summary>Copying... <progress max="375505392" value="97543282"></progress> 25%</summary>
    <dl>
      <dt>Transfer rate:</dt> <dd>452KB/s</dd>
      <dt>Local filename:</dt> <dd>/home/rpausch/raycd.m4v</dd>
      <dt>Remote filename:</dt> <dd>/var/www/lectures/raycd.m4v</dd>
      <dt>Duration:</dt> <dd>01:16:27</dd>
      <dt>Color profile:</dt> <dd>SD (6-1-6)</dd>
      <dt>Dimensions:</dt> <dd>320×240</dd>
    </dl>
  </details>
</section>
```

EXAMPLE 545

The following shows how a `<details>` element can be used to hide some controls by default:

```
<details>
  <summary><label for=fn>Name & Extension:</label></summary>
  <p><input type=text id=fn name=fn value="Pillar Magazine.pdf">
  <p><label><input type=checkbox name=ext checked> Hide extension</label>
</details>
```

One could use this in conjunction with other `details` in a list to allow the user to collapse a set of fields down to a small set of headings, with the ability to open each one.



In these examples, the summary really just summarizes what the controls can change, and not the actual values, which is less than ideal.

EXAMPLE 546

Because the open attribute is added and removed automatically as the user interacts with the control, it can be used in CSS to style the element differently based on its state. Here, a stylesheet is used to animate the color of the summary when the element is opened or closed:

```
<style>
  details > summary { transition: color 1s; color: black; }
  details[open] > summary { color: red; }
</style>
<details>
  <summary>Automated Status: Operational</summary>
  <p>Velocity: 12m/s</p>
  <p>Direction: North</p>
</details>
```

§ 4.11.2. The summary element**Categories:**

None.

Contexts in which this element can be used:

As the first child of a [<details>](#) element.

Content model:

Either: [phrasing content](#).

Or: one element of [heading content](#).

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

[‘button’](#).

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

Uses HTMLElement.

The [<summary>](#) element [represents](#) a summary, caption, or legend for the rest of the contents of the [<summary>](#) element’s parent details element, if any.

§ 4.11.3. The menu element**Categories:**

[Flow content](#).

Contexts in which this element can be used:

Where [flow content](#) is expected.

If the element’s type attribute is in the [popup menu](#) state: as the child of a [<menu>](#) element whose type attribute is in the [popup menu](#) state.

Content model:

If the element’s type attribute is in the [popup menu](#) state: in any order, zero or more [<menuitem>](#) elements, zero or more [<hr>](#) elements, zero or more [<menu>](#) elements whose type attributes are in the [popup menu](#) state, and zero or more [script-supporting elements](#).

Tag omission in text/html:

Neither tag is omissible

Content attributes:

Global attributes

type - Type of menu

label - User-visible label

Allowed ARIA role attribute values:

'menu' (default - *do not set*), 'directory', 'list', 'listbox', 'menubar', 'tablist', 'tabpanel' or 'tree'.

Allowed ARIA state and property attributes:

Global aria-* attributes

Any aria-* attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLMenuElement : HTMLElement {
    attribute DOMString type;
    attribute DOMString label;
};
```

The `<menu>` element represents a group of commands.

The **type** attribute is an [enumerated attribute](#) indicating the kind of menu being declared. The attribute has 1 state. The "context" keyword maps to the **popup menu** state, in which the element is declaring a context menu. The attribute may also be omitted. The *missing value default* is the [popup menu](#) state.

If a `<menu>` element's type attribute is in the [popup menu](#) state, then the element [represents](#) the commands of a popup menu, and the user can only examine and interact with the commands if that popup menu is activated through some other element via the [contextmenu](#) attribute.

The **label** attribute gives the label of the menu. It is used by user agents to display nested menus in the UI: a context menu containing another menu would use the nested menu's label attribute for the submenu's menu label. The label attribute must only be specified on `<menu>` elements whose parent element is a `<menu>` element whose type attribute is in the [popup menu](#) state.



A menu is a **currently relevant `<menu>` element** if it is the child of a currently relevant `<menu>` element, or if it is the [designated pop-up menu](#) of a `<button>` element that is not [inert](#), does not have a hidden attribute, and is not the descendant of an element with a hidden attribute.



A **menu construct** consists of an ordered list of zero or more **menu item constructs**, which can be any of:

- **Commands**, which can be marked as **default commands** (menuitem)
- **Separators** (hr)
- Other [menu constructs](#), each with an associated **submenu label**, which allows the list to be nested (menu)

To **build and show a menu** for a particular `<menu>` element *source* and with a particular element *subject* as a subject, the user agent must run the following steps:

1. Let *pop-up menu* be the [menu construct](#) created by the [build a menu construct](#) algorithm when passed the *source* element.
2. Display *pop-up menu* to the user, and let the algorithm that invoked this one continue.

If the user selects a [menu item construct](#) that corresponds to an element that still represents a [command](#) when the user selects it, then the user agent must invoke that command's [Action](#). If the command's [Action](#) is defined as [firing a click event](#), either directly or via the [run synthetic click activation steps](#) algorithm, then the `relatedTarget` attribute of that click event must be initialized to *subject*.

Pop-up menus must not, while being shown, reflect changes in the DOM. The menu is constructed from the DOM before being shown, and is then immutable.

To **build a menu construct** for an element *source*, the user agent must run the following steps, which return a [menu construct](#):

1. Let *generated menu* be an empty [menu construct](#).
2. Run the [menu item generator](#) steps for the `<menu>` element using *generated menu* as the output.

The **menu item generator** steps for a `<menu>` element using a specific [menu construct](#) *output* as output are as follows: For each child node of the menu in [tree order](#), run the appropriate steps from the following list:

↪ **If the child is a `<menuitem>` element that defines a command**

Append the [command](#) to *output*, respecting the command's [facets](#). If the `<menuitem>` element has a default attribute, mark the [command](#) as being a [default command](#).

↪ **If the child is an `<hr>` element**

Append a [separator](#) to *output*.

↪ **If the child is a `<menu>` element with no `label` attribute**

Append a [separator](#) to *output*, then run the [menu item generator](#) steps for this child `<menu>` element, using *output* as the output, then append another [separator](#) to *output*.

↪ **If the child is a `<menu>` element with a `label` attribute**

Let *submenu* be the result of running the [build a menu construct](#) steps for the child `<menu>` element. Then, append *submenu* to *output*, using the value of the child `<menu>` element's `label` attribute as the [submenu label](#).

↪ **Otherwise**

[Ignore](#) the child node.

3. Remove from *output* any [menu construct](#) whose [submenu label](#) is the empty string.
4. Remove from *output* any [menu item construct](#) representing a [command](#) whose [Label](#) is the empty string.
5. Collapse all sequences of two or more adjacent [separators](#) in *output* to a single [separator](#).
6. If the first [menu item construct](#) in *output* is a [separator](#), then remove it.
7. If the last [menu item construct](#) in *output* is a [separator](#), then remove it.
8. Return *output*.



The **type** IDL attribute must [reflect](#) the content attribute of the same name, [limited to only known values](#).

The **label** IDL attribute must [reflect](#) the content attribute of the same name.

§ 4.11.4. The `menuitem` element

Categories:

None.

Contexts in which this element can be used:

As a child of a `<menu>` element whose `type` attribute is in the [popup menu](#) state.

Content model:

[Nothing](#).

Tag omission in text/html:

No [end tag](#).

Content attributes:

[Global attributes](#)

`type` - Type of command

`label` - User-visible label

`icon` - Icon for the command

`disabled` Whether the command or control is disabled

`checked` Whether the command or control is checked

`radiogroup` Name of group of commands to treat as a radio button group

`default` - Mark the command as being a default command

Also, the [title](#) attribute has special semantics on this element.

Allowed [ARIA role attribute](#) values:

'[menuitem](#)' (default - *do not set*).

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

Any `aria-*` attributes [applicable to the allowed roles](#).

DOM interface:

```
interface HTMLMenuItemElement : HTMLElement {
  attribute DOMString type;
  attribute DOMString label;
  attribute DOMString icon;
  attribute boolean disabled;
  attribute boolean checked;
  attribute DOMString radiogroup;
  attribute boolean default;
};
```

The `<menuitem>` element represents a command that the user can invoke from a popup menu (a [context menu](#)).

A `<menuitem>` element that uses one or more of the `type`, `label`, `icon`, `disabled`, `checked`, and `radiogroup` attributes defines a new command.



The **type** attribute indicates the kind of command: either a normal command with an associated action, or a state or option that can be toggled, or a selection of one item from a list of items.

The attribute is an [enumerated attribute](#) with three keywords and states. The "**command**" keyword maps to the [Command](#) state, the "**checkbox**" keyword maps to the [Checkbox](#) state, and the "**radio**" keyword maps to the [Radio](#) state. The *missing value default* is the [Command](#) state.

The *Command* state

The element [represents](#) a normal command with an associated action.

The *Checkbox* state

The element [represents](#) a state or option that can be toggled.

The *Radio* state

The element [represents](#) a selection of one item from a list of items.

The **label** attribute gives the name of the command, as shown to the user. If the attribute is specified, it must have a value that is not the empty string.

The **icon** attribute gives a picture that represents the command. If the attribute is specified, the attribute's value must contain a [valid non-empty URL potentially surrounded by spaces](#). To obtain the [absolute URL](#) of the icon when the attribute's value is not the empty string, the attribute's value must be [resolved](#) relative to the element. When the attribute is absent, or its value is the empty string, or [parsing](#) its value fails, there is no icon.

The **disabled** attribute is a [boolean attribute](#) that, if present, indicates that the command is not available in the current state.

NOTE:

The distinction between `disabled` and `hidden` is subtle. A command would be disabled if, in the same context, it could be enabled if only certain aspects of the situation were changed. A command would be marked as hidden if, in that situation, the command will never be enabled. For example, in the context menu for a water faucet, the command "open" might be disabled if the faucet is already open, but the command "eat" would be marked hidden since the faucet could never be eaten.

The **checked** attribute is a [boolean attribute](#) that, if present, indicates that the command is selected. The attribute must be omitted unless the `type` attribute is in either the [Checkbox](#) state or the [Radio](#) state.

The **radiogroup** attribute gives the name of the group of commands that will be toggled when the command itself is toggled, for commands whose `type` attribute has the value "radio". The scope of the name is the child list of the parent element. The attribute must be omitted unless the `type` attribute is in the [Radio](#) state. When specified, the attribute's value must be a non-empty string.



The **title** attribute gives a hint describing the command, which might be shown to the user to help him.

The **default** attribute indicates, if present, that the command is the one that would have been invoked if the user had directly activated the menu's subject instead of using the menu. The `default` attribute is a [boolean attribute](#).



The **type** IDL attribute must [reflect](#) the content attribute of the same name, [limited to only known values](#).

The **label**, **icon**, **disabled**, **checked**, and **radiogroup**, and **default** IDL attributes must [reflect](#) the respective content attributes of the same name.



If the element's [Disabled State](#) is false (enabled) then the element's [activation behavior](#) depends on the element's `type` attribute, as follows:

↪ **If the `type` attribute is in the [Checkbox](#) state**

If the element has a `checked` attribute, the user agent must remove that attribute. Otherwise, the user agent must add a `checked` attribute, with the literal value "checked".

↪ **If the `type` attribute is in the [Radio](#) state**

If the element has a parent, then the user agent must walk the list of child nodes of that parent element, and for each node that is a `<menuitem>` element, if that element has a `radiogroup` attribute whose value exactly matches the current element's (treating missing `radiogroup` attributes as if they were the empty string), and has a `checked` attribute, must remove that attribute.

Then, the element's `checked` attribute must be set to the literal value "checked".

↪ **Otherwise**

The element's [activation behavior](#) is to do nothing.

NOTE:

Firing a synthetic `click` event at the element does not cause any of the actions described above to happen.

If the element's [Disabled State](#) is true (disabled) then the element has no [activation behavior](#).

NOTE:

The `<menuitem>` element is not rendered except as [part of a popup menu](#).

§ 4.11.5. Context menus

§ 4.11.5.1. Declaring a context menu

The **contextmenu** attribute gives the element's context menu. The value must be the [ID](#) of a [<menu>](#) element in the same [home subtree](#) whose type attribute is in the [popup menu](#) state.

⚠Warning! The [contextmenu](#) attribute is "at risk". If testing during the Candidate Recommendation phase does not identify at least two interoperable implementations in current shipping browsers of the [contextmenu](#) attribute it will be removed from the HTML 5.1 Specification.

NOTE:

When a user right-clicks on an element with a [contextmenu](#) attribute, the user agent will first fire a contextmenu event at the element, and then, if that event is not canceled, a show event at the [<menu>](#) element.

EXAMPLE 547

Here is an example of a context menu for an input control:

```
<form name="npc">
  <label>Character name: <input name=char type=text contextmenu=namemenu required></label>
  <menu type=context id=namemenu>
    <menuitem label="Pick random name" onclick="document.forms.npc.elements.char.value =
getRandomName()">
    <menuitem label="Prefill other fields based on name"
onclick="prefillFields(document.forms.npc.elements.char.value)">
  </menu>
</form>
```

This adds two items to the control's context menu, one called "Pick random name", and one called "Prefill other fields based on name". They invoke scripts that are not shown in the example above.

§ 4.11.5.2. Processing model

Each element has an **assigned context menu**, which can be null. If an element *A* has a [contextmenu](#) attribute, and there is an element with the ID given by *A*'s [contextmenu](#) attribute's value in *A*'s [home subtree](#), and the first such element in [tree order](#) is a [<menu>](#) element whose type attribute is in the [popup menu](#) state, then *A*'s [assigned context menu](#) is that element. Otherwise, if *A* has a parent element, then *A*'s [assigned context menu](#) is the [assigned context menu](#) of its parent element. Otherwise, *A*'s [assigned context menu](#) is null.

When an element's context menu is requested (e.g., by the user right-clicking the element, or pressing a context menu key), the user agent must apply the appropriate rules from the following list:

↪ If the user requested a context menu using a pointing device

The user agent must [fire](#) a [trusted](#) event with the name contextmenu, that bubbles and is cancelable, and that uses the MouseEvent interface, at the element for which the menu was requested. The context information of the event must be initialized to the same values as the last MouseEvent user interaction event that was fired as part of the gesture that was interpreted as a request for the context menu.

↪ Otherwise

The user agent must [fire a synthetic mouse event named contextmenu](#) that bubbles and is cancelable at the element for which the menu was requested.

NOTE:

Typically, therefore, the firing of the contextmenu event will be the default action of a mouseup or keyup event. The exact sequence of events is user agent-dependent, as it will vary based on platform conventions.

The default action of the contextmenu event depends on whether or not the element for which the menu was requested has a

non-null [assigned context menu](#) when the event dispatch has completed, as follows.

If the [assigned context menu](#) of the element for which the menu was requested is null, the default action must be for the user agent to show its default context menu, if it has one.

Otherwise, let *subject* be the element for which the menu was requested, and let *menu* be the [assigned context menu](#) of *target* immediately after the `contextmenu` event's dispatch has completed. The user agent must [fire a trusted](#) event with the name `show` at *menu*, using the `RelatedEvent` interface, with the `relatedTarget` attribute initialized to *subject*. The event must be cancelable.

If *this* event (the `show` event) is not canceled, then the user agent must [build and show](#) the menu for *menu* with *subject* as the subject.

The user agent may also provide access to its default context menu, if any, with the context menu shown. For example, it could merge the menu items from the two menus together, or provide the page's context menu as a submenu of the default menu. In general, user agents are encouraged to de-emphasize their own contextual menu items, so as to give the author's context menu the appearance of legitimacy — to allow documents to feel like "applications" rather than "mere Web pages".

User agents may provide means for bypassing the context menu processing model, ensuring that the user can always access the user agent's default context menus. For example, the user agent could handle right-clicks that have the Shift key depressed in such a way that it does not fire the `contextmenu` event and instead always shows the default context menu.



The `contextMenu` IDL attribute must [reflect](#) the `contextmenu` content attribute.

EXAMPLE 548

In this example, an image of cats is given a context menu with four possible commands:

```

<menu type="context" id="catsmenu">
  <menuitem label="Pet the kittens" onclick="kittens.pet()">
  <menuitem label="Cuddle with the kittens" onclick="kittens.cuddle()">
  <menu label="Feed the kittens">
    <menuitem label="Fish" onclick="kittens.feed(fish)">
    <menuitem label="Chicken" onclick="kittens.feed(chicken)">
  </menu>
</menu>
```

When a user of a mouse-operated visual Web browser right-clicks on the image, the browser might pop up a context menu like this:



When the user clicks the disclosure triangle, such a user agent would expand the context menu in place, to show the browser's own commands:



4.11.5.3. The *RelatedEvent* interfaces

```
[Constructor(DOMString type, optional RelatedEventInit eventInitDict)]
interface RelatedEvent : Event {
  readonly attribute EventTarget? relatedTarget;
};

dictionary RelatedEventInit : EventInit {
  EventTarget? relatedTarget;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

event . ***relatedTarget***

Returns the other event target involved in this event. For example, when a `show` event fires on a `<menu>` element, the other event target involved in the event would be the element for which the menu is being shown.

The **relatedTarget** attribute must return the value it was initialized to. When the object is created, this attribute must be initialized to null. It represents the other event target that is related to the event.

§ 4.11.6. Commands

§ 4.11.6.1. Facets

A **command** is the abstraction behind menu items, buttons, and links. Once a command is defined, other parts of the interface can refer to the same command, allowing many access points to a single feature to share facets such as the [Disabled State](#).

Commands are defined to have the following **facets**:

Label

The name of the command as seen by the user.

Access Key

A key combination selected by the user agent that triggers the command. A command might not have an Access Key.

Hidden State

Whether the command is hidden or not (basically, whether it should be shown in menus).

Disabled State

Whether the command is relevant and can be triggered or not.

Action

The actual effect that triggering the command will have. This could be a scripted event handler, a [URL](#) to which to [navigate](#), or a form submission.

User agents may expose the [commands](#) that match the following criteria:

- The [Hidden State](#) facet is false (visible)
- The element is [in a Document](#) that has an associated [browsing context](#).
- Neither the element nor any of its ancestors has a hidden attribute specified.
- The element is not a `<menuitem>` element, or it is a child of a [currently relevant menu element](#), or it has an [Access Key](#).

User agents are encouraged to do this especially for commands that have [Access Keys](#), as a way to advertise those keys to the user.

EXAMPLE 549

For example, such commands could be listed in the user agent's menu bar.

§ 4.11.6.2. Using the `<a>` element to define a command

An `<a>` element with an [href](#) attribute defines a command.

The [Label](#) of the command is the string given by the element's [textContent](#) IDL attribute.

The [Access Key](#) of the command is the element's [assigned access key](#), if any.

The [Hidden State](#) of the command is true (hidden) if the element has a hidden attribute, and false otherwise.

The [Disabled State](#) facet of the command is true if the element or one of its ancestors is [inert](#), and false otherwise.

The [Action](#) of the command, if the element has a defined [activation behavior](#), is to [run synthetic click activation steps](#) on the element. Otherwise, it is just to [fire a click event](#) at the element.

§ 4.11.6.3. Using the `<button>` element to define a command

A `<button>` element always [defines a command](#).

The [Label](#), [Access Key](#), [Hidden State](#), and [Action](#) facets of the command are determined as for `<a>` elements (see the previous section).

The [Disabled State](#) of the command is true if the element or one of its ancestors is [inert](#), or if the element's disabled state is set, and false otherwise.

4.11.6.4. Using the `<input>` element to define a command

An `<input>` element whose type attribute is in one of the [submit button](#), [reset button](#), [Image Button](#), [Button](#), [Radio Button](#), or [Checkbox](#) states [defines a command](#).

The [Label](#) of the command is determined as follows:

- If the type attribute is in one of the [submit button](#), [reset button](#), [Image Button](#), or [Button](#) states, then the [Label](#) is the string given by the [value](#) attribute, if any, and a user agent-dependent, locale-dependent value that the user agent uses to label the button itself if the attribute is absent.
- Otherwise, if the element is a [labeled control](#), then the [Label](#) is the string given by the [textContent](#) of the first `<label>` element in [tree order](#) whose [labeled control](#) is the element in question. (In DOM terms, this is the string given by `element.labels[0].textContent`.)
- Otherwise, if the [value](#) attribute is present, then the [Label](#) is the value of that attribute.
- Otherwise, the [Label](#) is the empty string.

The [Access Key](#) of the command is the element's [assigned access key](#), if any.

The [Hidden State](#) of the command is true (hidden) if the element has a [hidden](#) attribute, and false otherwise.

The [Disabled State](#) of the command is true if the element or one of its ancestors is [inert](#), or if the element's disabled state is set, and false otherwise.

The [Action](#) of the command, if the element has a defined [activation behavior](#), is to [run synthetic click activation steps](#) on the element. Otherwise, it is just to [fire a click event](#) at the element.

4.11.6.5. Using the `<option>` element to define a command

An `<option>` element with an ancestor `<select>` element and either no [value](#) attribute or a [value](#) attribute that is not the empty string defines a command.

The [Label](#) of the command is the value of the `<option>` element's [label](#) attribute, if there is one, or else the value of `<option>` element's [textContent](#) IDL attribute, with [leading and trailing whitespace stripped](#), and with any sequences of two or more [space characters](#) replaced by a single U+0020 SPACE character.

The [Access Key](#) of the command is the element's [assigned access key](#), if any.

The [Hidden State](#) of the command is true (hidden) if the element has a [hidden](#) attribute, and false otherwise.

The [Disabled State](#) of the command is true if the element is disabled, or if its nearest ancestor `<select>` element is disabled, or if it or one of its ancestors is [inert](#), and false otherwise.

If the option's nearest ancestor `<select>` element has a [multiple](#) attribute, the [Action](#) of the command is to [pick](#) the `<option>` element. Otherwise, the [Action](#) is to [toggle](#) the `<option>` element.

4.11.6.6. Using the `<menuitem>` element to define a command

A `<menuitem>` element always [defines a command](#).

The [Label](#) of the command is the value of the element's [label](#) attribute, if there is one, or the empty string if it doesn't.

The [Access Key](#) of the command is the element's [assigned access key](#), if any.

The [Hidden State](#) of the command is true (hidden) if the element has a `hidden` attribute, and false otherwise.

The [Disabled State](#) of the command is true if the element or one of its ancestors is [inert](#), or if the element has a `disabled` attribute, and false otherwise.

The [Action](#) of the command, if the element has a defined [activation behavior](#), is to [run synthetic click activation steps](#) on the element. Otherwise, it is just to [fire a click event](#) at the element.

§ 4.11.6.7. Using the `accesskey` attribute on a `<label>` element to define a command

A `<label>` element that has an [assigned access key](#) and a [labeled control](#) and whose [labeled control](#) defines a command, itself defines a command.

The [Label](#) of the command is the string given by the element's [textContent](#) IDL attribute.

The [Access Key](#) of the command is the element's [assigned access key](#).

The [Hidden State](#), [Disabled State](#), and [Action](#) facets of the command are the same as the respective facets of the element's [labeled control](#).

§ 4.11.6.8. Using the `accesskey` attribute on a `<legend>` element to define a command

A `<legend>` element that has an [assigned access key](#) and is a child of a `<fieldset>` element that has a descendant that is not a descendant of the `<legend>` element and is neither a `<label>` element nor a `legend` element but that defines a command, itself defines a command.

The [Label](#) of the command is the string given by the element's [textContent](#) IDL attribute.

The [Access Key](#) of the command is the element's [assigned access key](#).

The [Hidden State](#), [Disabled State](#), and [Action](#) facets of the command are the same as the respective facets of the first element in [tree order](#) that is a descendant of the parent of the `<legend>` element that defines a command but is not a descendant of the `<legend>` element and is neither a `label` nor a `<legend>` element.

§ 4.11.6.9. Using the `accesskey` attribute to define a command on other elements

An element that has an [assigned access key](#) defines a command.

If one of the earlier sections that define elements that [define commands](#) define that this element defines a command, then that section applies to this element, and this section does not. Otherwise, this section applies to that element.

The [Label](#) of the command depends on the element. If the element is a [labeled control](#), the [textContent](#) of the first `<label>` element in [tree order](#) whose [labeled control](#) is the element in question is the [Label](#) (in DOM terms, this is the string given by `element.labels[0].textContent`). Otherwise, the [Label](#) is the [textContent](#) of the element itself.

The [Access Key](#) of the command is the element's [assigned access key](#).

The [Hidden State](#) of the command is true (hidden) if the element has a `hidden` attribute, and false otherwise.

The [Disabled State](#) of the command is true if the element or one of its ancestors is [inert](#), and false otherwise.

The [Action](#) of the command is to run the following steps:

1. Run the [focusing steps](#) for the element.
2. If the element has a defined [activation behavior](#), [run synthetic click activation steps](#) on the element.
3. Otherwise, if the element does not have a defined [activation behavior](#), [fire a click event](#) at the element.

§ 4.12. Scripting

Scripts allow authors to add interactivity to their documents.

Authors are encouraged to use declarative alternatives to scripting where possible, as declarative mechanisms are often more maintainable, and many users disable scripting.

EXAMPLE 550

For example, instead of using script to show or hide a section to show more details, the `<details>` element could be used.

Authors are also encouraged to make their applications degrade gracefully in the absence of scripting support.

EXAMPLE 551

For example, if an author provides a link in a table header to dynamically resort the table, the link could also be made to function without scripts by requesting the sorted table from the server.

§ 4.12.1. The `script` element

Categories:

[Metadata content.](#)
[Flow content.](#)
[Phrasing content.](#)
[Script-supporting element.](#)

Contexts in which this element can be used:

Where [metadata content](#) is expected.
Where [phrasing content](#) is expected.
Where [script-supporting elements](#) are expected.

Content model:

If there is no `src` attribute, depends on the value of the `type` attribute, but must match [script content restrictions](#).
If there *is* a `src` attribute, the element must be either empty or contain only [script documentation](#) that also matches [script content restrictions](#).

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)
[src](#) - Address of the resource
[type](#) - Type of embedded resource
[charset](#) - Character encoding of the external script resource
[async](#) - Execute script [in parallel](#)
[defer](#) - Defer script execution
[crossorigin](#) - How the element handles crossorigin requests
[nonce](#) - Cryptographic nonce used in *Content Security Policy* checks [\[CSP3\]](#)

Allowed [ARIA role attribute](#) values:

None

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

[DOM interface](#):

```
interface HTMLScriptElement : HTMLElement {
  attribute DOMString src;
  attribute DOMString type;
  attribute DOMString charset;
  attribute boolean async;
  attribute boolean defer;
  attribute DOMString? crossOrigin;
  attribute DOMString text;
  attribute DOMString nonce;
};
```

The `<script>` element allows authors to include dynamic script and data blocks in their documents. The element does not [represent](#) content for the user.

The **type** attribute allows customization of the type of script represented:

- Omitting the attribute, or setting it to a [JavaScript MIME type](#), means that the script is a [classic script](#), to be interpreted according to the JavaScript [Script](#) top-level production. Classic scripts are affected by the [charset](#), [async](#), and [defer](#) attributes. Authors should omit the attribute, instead of redundantly giving a [JavaScript MIME type](#).
- Setting the attribute to any other value means that the script is a **data block**, which is not processed. None of the `<script>` attributes (except **type** itself) have any effect on [data blocks](#). Authors must use a [valid MIME type](#) that is not a [JavaScript MIME type](#) to denote [data blocks](#).

NOTE:

The requirement that [data blocks](#) must be denoted using a [valid MIME type](#) is in place to avoid potential future collisions. If this specification ever adds additional types of `<script>`, they will be triggered by setting the **type** attribute to something which is not a MIME type. By using a valid MIME type now, you ensure that your data block will not ever be reinterpreted as a different script type, even in future user agents.

[Classic scripts](#) may either be embedded inline or may be imported from an external file using the **src** attribute, which if specified gives the [URL](#) of the external script resource to use. If **src** is specified, it must be a [valid non-empty URL potentially surrounded by spaces](#). The contents of inline `<script>` elements, or the external script resource, must conform with the requirements of the JavaScript specification's [Script](#) production for [classic scripts](#). [\[ECMA-262\]](#)

When used to include [data blocks](#), the data must be embedded inline, the format of the data must be given using the **type** attribute, and the contents of the `<script>` element must conform to the requirements defined for the format used. The [src](#), [charset](#), [async](#), [defer](#), [crossorigin](#), and [nonce](#) attributes must not be specified.

The **charset** attribute gives the character encoding of the external script resource. The attribute must not be specified if the [src](#) attribute is not present, or if the script is not a [classic script](#). If the attribute is set, its value must be an [ASCII case-insensitive](#) match for one of the [labels](#) of an [encoding](#), and must specify the same [encoding](#) as the **charset** parameter of the [Content-Type metadata](#) of the external file, if any. [\[ENCODING\]](#)

The **async** and **defer** attributes are [boolean attributes](#) that indicate how the script should be executed. [Classic scripts](#) may specify [defer](#) or [async](#).

There are several possible modes that can be selected using these attributes, and depending on the script's **type**.

For [classic scripts](#), if the [async](#) attribute is present, then the classic script will be fetched [in parallel](#) to parsing and evaluated as soon as it is available (potentially before parsing completes). If the [async](#) attribute is not present but the [defer](#) attribute is present, then the [classic script](#) will be fetched [in parallel](#) and evaluated when the page has finished parsing. If neither attribute is present, then the script is fetched and evaluated immediately, blocking parsing until these are both complete.

This is all summarized in the following schematic diagram:

**NOTE:**

The exact processing details for these attributes are, for mostly historical reasons, somewhat non-trivial, involving a number of aspects of HTML. The implementation requirements are therefore by necessity scattered throughout the specification. The algorithms below (in this section) describe the core of this processing, but these algorithms reference and are referenced by the parsing rules for `<script>` start and end tags in HTML, [in foreign content](#), and in XML, the rules for the `document.write()` method, the handling of [scripting](#), etc.

The [defer](#) attribute may be specified even if the [async](#) attribute is specified, to cause legacy Web browsers that only support [defer](#) (and not [async](#)) to fall back to the [defer](#) behavior instead of the blocking behavior that is the default.

The [crossorigin](#) attribute is a [CORS settings attribute](#). For [classic scripts](#), it controls whether error information will be exposed, when the script is obtained from other [origins](#).

The [nonce](#) attribute represents a cryptographic nonce ("number used once") which can be used by *Content Security Policy* to determine whether or not the script specified by an element will be executed. The value is text. [\[CSP3\]](#)

Changing the [src](#), [type](#), [charset](#), [async](#), [defer](#), [crossorigin](#), and [nonce](#) attributes dynamically has no direct effect; these attributes are only used at specific times described below.

The IDL attributes [src](#), [type](#), [charset](#), [defer](#), and [nonce](#), must each [reflect](#) the respective content attributes of the same name.

The [crossOrigin](#) IDL attribute must [reflect](#) the [crossorigin](#) content attribute.

The [async](#) IDL attribute controls whether the element will execute [in parallel](#) or not. If the element's "non-blocking" flag is set, then, on getting, the [async](#) IDL attribute must return true, and on setting, the "non-blocking" flag must first be unset, and then the content attribute must be removed if the IDL attribute's new value is false, and must be set to the empty string if the IDL attribute's new value is true. If the element's "non-blocking" flag is *not* set, the IDL attribute must [reflect](#) the [async](#) content attribute.

This definition is non-normative. Implementation requirements are given below this definition.

`script . text` [= value]

Returns the [child text content](#) of the element.

Can be set, to replace the element's children with the given value.

The IDL attribute [text](#) must return the [child text content](#) of the `<script>` element. On setting, it must act the same way as the [textContent](#) IDL attribute.

NOTE:

When inserted using the `document.write()` method, `<script>` elements execute (typically blocking further script execution or HTML parsing), but when inserted using `innerHTML` and `outerHTML` attributes, they do not execute at all.

EXAMPLE 552

In this example, two `<script>` elements are used. One embeds an external [classic script](#), and the other includes some data as a [data block](#).

```
<script src="game-engine.js"></script>
<script type="text/x-game-map">
.....U.....e
o.....A....e
....A....AAA...e
.A..AAA...AAAAA...e
</script>
```

The data in this case might be used by the script to generate the map of a video game. The data doesn't have to be used that way, though; maybe the map data is actually embedded in other parts of the page's markup, and the data block here is just used by the site's search engine to help users who are looking for particular features in their game maps.

EXAMPLE 553

The following sample shows how a `<script>` element can be used to define a function that is then used by other parts of the document, as part of a [classic script](#). It also shows how a `<script>` element can be used to invoke script while the document is being parsed, in this case to initialize the form's output.

```
<script>
function calculate(form) {
  var price = 52000;
  if (form.elements.brakes.checked)
    price += 1000;
  if (form.elements.radio.checked)
    price += 2500;
  if (form.elements.turbo.checked)
    price += 5000;
  if (form.elements.sticker.checked)
    price += 250;
  form.elements.result.value = price;
}
</script>
<form name="pricercalc" onsubmit="return false" onchange="calculate(this)">
  <fieldset>
    <legend>Work out the price of your car</legend>
    <p>Base cost: £52000.</p>
    <p>Select additional options:</p>
    <ul>
      <li><label><input type="checkbox" name="brakes"> Ceramic brakes (£1000)</label></li>
      <li><label><input type="checkbox" name="radio"> Satellite radio (£2500)</label></li>
      <li><label><input type="checkbox" name="turbo"> Turbo charger (£5000)</label></li>
      <li><label><input type="checkbox" name="sticker"> "XZ" sticker (£250)</label></li>
    </ul>
    <p>Total: £<output name="result"></output></p>
  </fieldset>
  <script>
    calculate(document.forms.pricercalc);
  </script>
</form>
```

4.12.1.1. Processing model

A `<script>` element has several associated pieces of state.

The first is a flag indicating whether or not the script block has been **"already started"**. Initially, `<script>` elements must have this flag unset (script blocks, when created, are not "already started"). The [cloning steps](#) for `<script>` elements must set

the "already started" flag on the copy if it is set on the element being cloned.

The second is a flag indicating whether the element was **"parser-inserted"**. Initially, `<script>` elements must have this flag unset. It is set by the [HTML parser](#) and the [XML parser](#) on `<script>` elements they insert and affects the processing of those elements.

The third is a flag indicating whether the element will **"non-blocking"**. Initially, `<script>` elements must have this flag set. It is unset by the [HTML parser](#) and the [XML parser](#) on `<script>` elements they insert. In addition, whenever a `<script>` element whose **"non-blocking"** flag is set has an `async` content attribute added, the element's **"non-blocking"** flag must be unset.

The fourth is a flag indicating whether or not the script block is **"ready to be parser-executed"**. Initially, `<script>` elements must have this flag unset (script blocks, when created, are not "ready to be parser-executed"). This flag is used only for elements that are also **"parser-inserted"**, to let the parser know when to execute the script.

The fifth is **the script's type**, which is "classic". It is determined when the script is [prepared](#), based on the `type` attribute of the element at that time. Initially, `<script>` elements must have this flag unset.

The sixth is a flag indicating whether or not the script is **from an external file**. It is determined when the script is [prepared](#), based on the `src` attribute of the element at that time.

Finally, a `<script>` element has **the script's script**, which is a `<script>` resulting from [preparing](#) the element. This is set asynchronously after the [classic script](#) is fetched. Once it is set, either to a `<script>` in the case of success or to null in the case of failure, the fetching algorithms will note that **the script is ready**, which can trigger other actions. The user agent must [delay the load event](#) of the element's [node document](#) until [the script is ready](#).

When a `<script>` element that is not marked as being **"parser-inserted"** experiences one of the events listed in the following list, the user agent must [immediately prepare](#) the `<script>` element:

- The `<script>` element gets [inserted into a document](#), at the time the node is inserted according to the DOM, after any other `<script>` elements inserted at the same time that are earlier in the Document in [tree order](#).
- The `<script>` element is [in a Document](#) and a node or document fragment is [inserted](#) into the `<script>` element, after any `<script>` elements [inserted](#) at that time.
- The `<script>` element is [in a Document](#) and has a `src` attribute set where previously the element had no such attribute.

To **prepare a script**, the user agent must act as follows:

1. If the `<script>` element is marked as having **"already started"**, then the user agent must abort these steps at this point. The script is not executed.
2. If the element has its **"parser-inserted"** flag set, then set *was-parser-inserted* to true and unset the element's **"parser-inserted"** flag. Otherwise, set *was-parser-inserted* to false.

NOTE:

This is done so that if parser-inserted `<script>` elements fail to run when the parser tries to run them, e.g., because they are empty or specify an unsupported scripting language, another script can later mutate them and cause them to run again.

3. If *was-parser-inserted* is true and the element does not have an `async` attribute, then set the element's **"non-blocking"** flag to true.

NOTE:

This is done so that if a parser-inserted `<script>` element fails to run when the parser tries to run it, but it is later executed after a script dynamically updates it, it will execute in a non-blocking fashion even if the `async` attribute isn't set.

4. If the element has no `src` attribute, and its child nodes, if any, consist only of comment nodes and empty `Text` nodes, then abort these steps at this point. The script is not executed.
5. If the element is not [in a Document](#), then the user agent must abort these steps at this point. The script is not executed.

6. If either:

- the `<script>` element has a `type` attribute and its value is the empty string, or
- the `<script>` element has no `type` attribute but it has a `language` attribute and *that* attribute's value is the empty string, or
- the `<script>` element has neither a `type` attribute nor a `language` attribute, then

...let *the script block's type string* for this `<script>` element be `"text/javascript"`.

Otherwise, if the `<script>` element has a `type` attribute, let *the script block's type string* for this `<script>` element be the value of that attribute with any leading or trailing sequences of [space characters](#) removed.

Otherwise, the element has a non-empty `language` attribute; let *the script block's type string* for this `<script>` element be the concatenation of the string `"text/"` followed by the value of the `language` attribute.

NOTE:

The `language` attribute is never conforming, and is always ignored if there is a `type` attribute present.

Determine *the script's type* as follows:

- If *the script block's type string* is an [ASCII case-insensitive](#) match for any [JavaScript MIME type](#), *the script's type* is `"classic"`.
- If neither of the above conditions are true, then abort these steps at this point. No script is executed.

7. If *was-parser-inserted* is true, then flag the element as `"parser-inserted"` again, and set the element's `"non-blocking"` flag to false.

8. The user agent must set the element's `"already started"` flag.

9. If the element is flagged as `"parser-inserted"`, but the element's `node document` is not the `Document` of the parser that created the element, then abort these steps.

10. If [scripting is disabled](#) for the `<script>` element, then abort these steps at this point. The script is not executed.

NOTE:

The definition of [scripting is disabled](#) means that, amongst others, the following scripts will not execute: scripts in `XMLHttpRequest`'s `responseXML` documents, scripts in `DOMParser`-created documents, scripts in documents created by `XSLTProcessor`'s `transformToDocument` feature, and scripts that are first inserted by a script into a `Document` that was created using the `createDocument()` API. [\[XHR\]](#) [\[DOM-Parsing\]](#) [\[DOM\]](#)

11. If the `<script>` element does not have a `src` content attribute, and the [Should element's inline behavior be blocked by Content Security Policy?](#) algorithm returns "Blocked" when executed upon the `<script>` element, `"script"`, and the `<script>` element's `child text content`, then abort these steps. The script is not executed. [\[CSP3\]](#)

12. If the `<script>` element has an `event` attribute and a `for` attribute, and *the script's type* is `"classic"`, then run these substeps:

1. Let *for* be the value of the `for` attribute.
2. Let *event* be the value of the `event` attribute.
3. [Strip leading and trailing whitespace](#) from *event* and *for*.
4. If *for* is not an [ASCII case-insensitive](#) match for the string `"window"`, then the user agent must abort these steps at this point. The script is not executed.
5. If *event* is not an [ASCII case-insensitive](#) match for either the string `"onload"` or the string `"onload()"`, then the user agent must abort these steps at this point. The script is not executed.

13. If the `<script>` element has a `charset` attribute, then let *encoding* be the result of [getting an encoding](#) from the value of the `charset` attribute.

If the `<script>` element does not have a `charset` attribute, or if [getting an encoding](#) failed, let *encoding* be the same as [the encoding of the document itself](#).

14. Let *CORS setting* be the current state of the element's `crossorigin` content attribute.

15. If the `<script>` element has a `nonce` attribute, then let *cryptographic nonce* be that attribute's value.

Otherwise, let *cryptographic nonce* be the empty string.

16. Let *parser state* be "parser-inserted" if the `<script>` element has been flagged as "parser-inserted", and "not parser-inserted" otherwise.

17. Let *settings* be the element's [node document](#)'s [Window](#) object's [environment settings object](#).

18. If the element has a `src` content attribute, run these substeps:

1. Let *src* be the value of the element's `src` attribute.
2. If *src* is the empty string, [queue a task](#) to [fire a simple event](#) named error at the element, and abort these steps.
3. Set the element's [from an external file](#) flag.
4. [Parse](#) *src* relative to the element's [node document](#).
5. If the previous step failed, [queue a task](#) to [fire a simple event](#) named error at the element, and abort these steps.

Otherwise, let *url* be the [resulting URL record](#).

6. Switch on [the script's type](#):

↪ `"classic"`

[Fetch a classic script](#) given *url*, *CORS setting*, *cryptographic nonce*, *parser state*, *settings*, and *encoding*.

When the chosen algorithm asynchronously completes, set [the script's script](#) to the result. At that time, [the script is ready](#).

For performance reasons, user agents may start fetching the classic script (as defined above) as the `src` attribute is set, instead, in the hope that the element will be [inserted into the document](#) (and that the `crossorigin` attribute won't change value in the meantime). Either way, once the element is [inserted into the document](#), the load must have started as described in this step. If the UA performs such prefetching, but the element is never inserted in the document, or the `src` attribute is dynamically changed, or the `crossorigin` attribute is dynamically changed, then the user agent will not execute the script so obtained, and the fetching process will have been effectively wasted.

19. If the element does not have a `src` content attribute, run these substeps:

1. Let *source text* be the value of the `text` IDL attribute.
2. Switch on [the script's type](#):

↪ `"classic"`

1. Let *script* be the result of [creating a classic script](#) using *source text* and *settings*.
2. Set [the script's script](#) to *script*.
3. [The script is ready](#).

20. Then, follow the first of the following options that describes the situation:

↪

the script's type	src present?	defer present?	async present?	other conditions
<code>"classic"</code>	yes	yes	no	element flagged as "parser-inserted"

Add the element to the end of the [list of scripts that will execute when the document has finished parsing](#)

associated with the [Document](#) of the parser that created the element.

When the [the script is ready](#), set the element's ["ready to be parser-executed"](#) flag. The parser will handle executing the script.



the script's type	src present?	defer present?	async present?	other conditions
"classic"	yes	no	no	element flagged as "parser-inserted"

The element is the [pending parsing-blocking script](#) of the [Document](#) of the parser that created the element.

(There can only be one such script per [Document](#) at a time.)

When [the script is ready](#), set the element's ["ready to be parser-executed"](#) flag. The parser will handle executing the script.



the script's type	src present?	defer present?	async present?	other conditions
"classic"	yes	yes or no	no	"non-blocking" flag not set on element

Add the element to the end of the [list of scripts that will execute in order as soon as possible](#) associated with the [node document](#) of the [<script>](#) element at the time the [prepare a script](#) algorithm started.

When [the script is ready](#), run the following steps:

1. If the element is not now the first element in the [list of scripts that will execute in order as soon as possible](#) to which it was added above, then mark the element as ready but abort these steps without executing the script yet.
2. *Execution*: [Execute the script block](#) corresponding to the first script element in this [list of scripts that will execute in order as soon as possible](#).
3. Remove the first element from this [list of scripts that will execute in order as soon as possible](#).
4. If this [list of scripts that will execute in order as soon as possible](#) is still not empty and the first entry has already been marked as ready, then jump back to the step labeled *Execution*.



the script's type	src present?	defer present?	async present?	other conditions
"classic"	yes	yes or no	yes or no	n/a

The element must be added to the [set of scripts that will execute as soon as possible](#) of the [node document](#) of the [<script>](#) element at the time the [prepare a script](#) algorithm started.

When [the script is ready](#), [execute the script block](#) and then remove the element from the [set of scripts that will execute as soon as possible](#).



the script's type	src present?	defer present?	async present?	other conditions
"classic"	no	yes or no	yes or no	All of the following: ◦ element flagged as "parser-inserted" ◦ an XML parser or an HTML parser whose script nesting level is not greater than one created the <script> ◦ the Document of the XML parser or HTML parser that created the <script> has a style sheet that is blocking scripts

The element is the [pending parsing-blocking script](#) of the [Document](#) of the parser that created the element.
 (There can only be one such script per [Document](#) at a time.)
 Set the element's ["ready to be parser-executed"](#) flag. The parser will handle executing the script.

↪ Otherwise

[Immediately execute the script block](#), even if other scripts are already executing.

The **pending parsing-blocking script** of a [Document](#) is used by the [Document](#)'s parser(s).

NOTE:

If a `<script>` element that blocks a parser gets moved to another [Document](#) before it would normally have stopped blocking that parser, it nonetheless continues blocking that parser until the condition that causes it to be blocking the parser no longer applies (e.g., if the script is a [pending parsing-blocking script](#) because there was [a style sheet that is blocking scripts](#) when it was parsed, but then the script is moved to another [Document](#) before the style sheet loads, the script still blocks the parser until the style sheets are all loaded, at which time the script executes and the parser is unblocked).

When the user agent is required to **execute a script block**, it must run the following steps:

1. If the element is flagged as ["parser-inserted"](#), but the element's [node document](#) is not the [Document](#) of the parser that created the element, then abort these steps.
2. If [the script's script](#) is null, [fire a simple event](#) named error at the element, and abort these steps.
3. If the script is [from an external file](#), then increment the [ignore-destructive-writes counter](#) of the `<script>` element's [node document](#). Let *neutralized doc* be that [Document](#).
4. Let *old script element* be the value to which the `<script>` element's [node document](#)'s `currentScript` object was most recently set.
5. Switch on [the script's type](#):

↪ `"classic"`

1. Set the `<script>` element's [node document](#)'s `currentScript` attribute to the `<script>` element.

NOTE:

This does not use the [in a document](#) check, as the `<script>` element could have been removed from the document prior to execution, and in that scenario `currentScript` still needs to point to it.

2. [Run the classic script](#) given by [the script's script](#).

6. Set the `<script>` element's [node document](#)'s `currentScript` object to *old script element*.
7. Decrement the [ignore-destructive-writes counter](#) of *neutralized doc*, if it was incremented in the earlier step.
8. If [the script's type](#) is "classic" and the script is [from an external file](#), [fire a simple event](#) named load at the `<script>` element.

Otherwise [queue a task](#) to [fire a simple event](#) named load at the `<script>` element.

4.12.1.2. Scripting languages

A **JavaScript MIME type** is a [MIME type](#) string that is one of the following and refers to JavaScript: [\[ECMA-262\]](#)

- application/ecmascript
- application/javascript
- application/x-ecmascript
- application/x-javascript
- text/ecmascript
- text/javascript
- text/javascript1.0
- text/javascript1.1
- text/javascript1.2
- text/javascript1.3

- `text/javascript1.4`
- `text/javascript1.5`
- `text/jscript`
- `text/livescript`
- `text/x-ecmascript`
- `text/x-javascript`

User agents must recognize all [JavaScript MIME types](#).

User agents may support other [MIME types](#) for other languages, but must not support other [MIME types](#) for the languages in the list above. User agents are not required to support JavaScript. The processing model for languages other than JavaScript is outside the scope of this specification.

The following [MIME types](#) (with or without parameters) must not be interpreted as scripting languages:

- ``text/plain``
- ``text/xml``
- ``application/octet-stream``
- ``application/xml``

NOTE:

These types are explicitly listed here because they are poorly-defined types that are nonetheless likely to be used as formats for data blocks, and it would be problematic if they were suddenly to be interpreted as script by a user agent.

When examining types to determine if they represent supported languages, user agents must not ignore MIME parameters. Types are to be compared including all parameters.

NOTE:

For example, types that include the `charset` parameter will not be recognized as referencing any of the scripting languages listed above.

4.12.1.3. Restrictions for contents of `<script>` elements

NOTE:

The easiest and safest way to avoid the rather strange restrictions described in this section is to always escape `"<!--"` as `"<\!--"`, `"<script"` as `"<\script"`, and `"</script"` as `"</\script"` when these sequences appear in literals in scripts (e.g., in strings, regular expressions, or comments), and to avoid writing code that uses such constructs in expressions. Doing so avoids the pitfalls that the restrictions in this section are prone to triggering: namely, that, for historical reasons, parsing of `<script>` blocks in HTML is a strange and exotic practice that acts unintuitively in the face of these sequences.

The [textContent](#) of a `<script>` element must match the script production in the following ABNF, the character set for which is Unicode. [\[ABNF\]](#)

```

script      = outer *( comment-open inner comment-close outer )

outer       = < any string that doesn't contain a substring that matches not-in-outer >
not-in-outer = comment-open
inner       = < any string that doesn't contain a substring that matches not-in-inner >
not-in-inner = comment-close / script-open

comment-open = "<!--"
comment-close = "-->"
script-open  = "<" s c r i p t tag-end

s           = %x0053 ; U+0053 LATIN CAPITAL LETTER S
s           =/ %x0073 ; U+0073 LATIN SMALL LETTER S
c           = %x0043 ; U+0043 LATIN CAPITAL LETTER C
c           =/ %x0063 ; U+0063 LATIN SMALL LETTER C
r           = %x0052 ; U+0052 LATIN CAPITAL LETTER R
r           =/ %x0072 ; U+0072 LATIN SMALL LETTER R
i           = %x0049 ; U+0049 LATIN CAPITAL LETTER I
i           =/ %x0069 ; U+0069 LATIN SMALL LETTER I
p           = %x0050 ; U+0050 LATIN CAPITAL LETTER P
p           =/ %x0070 ; U+0070 LATIN SMALL LETTER P
t           = %x0054 ; U+0054 LATIN CAPITAL LETTER T
t           =/ %x0074 ; U+0074 LATIN SMALL LETTER T

tag-end     = %x0009 ; U+0009 CHARACTER TABULATION (tab)
tag-end     =/ %x000A ; U+000A LINE FEED (LF)
tag-end     =/ %x000C ; U+000C FORM FEED (FF)
tag-end     =/ %x0020 ; U+0020 SPACE
tag-end     =/ %x002F ; U+002F SOLIDUS (/)
tag-end     =/ %x003E ; U+003E GREATER-THAN SIGN (>)

```

When a `<script>` element contains [script documentation](#), there are further restrictions on the contents of the element, as described in the section below.

EXAMPLE 554

The following script illustrates this issue. Suppose you have a script that contains a string, as in:

```
var example = 'Consider this string: <!-- <script>';
console.log(example);
```

If one were to put this string directly in a `<script>` block, it would violate the restrictions above:

```
<script>
  var example = 'Consider this string: <!-- <script>';
  console.log(example);
</script>
```

The bigger problem, though, and the reason why it would violate those restrictions, is that actually the script would get parsed weirdly: *the script block above is not terminated*. That is, what looks like a `"</script>"` end tag in this snippet is actually still part of the `<script>` block. The script doesn't execute (since it's not terminated); if it somehow were to execute, as it might if the markup looked as follows, it would fail because the script is not valid JavaScript:

```
<script>
  var example = 'Consider this string: <!-- <script>';
  console.log(example);
</script>
<!-- despite appearances, this is actually part of the script still! -->
<script>
  ... // this is the same script block still...
</script>
```

What is going on here is that for legacy reasons, `"<!--"` and `"<script"` strings in `<script>` elements in HTML need to be balanced in order for the parser to consider closing the block.

By escaping the problematic strings as mentioned at the top of this section, the problem is avoided entirely:

```
<script>
  var example = 'Consider this string: <\!-- <\script>';
  console.log(example);
</script>
<!-- this is just a comment between script blocks -->
<script>
  ... // this is a new script block
</script>
```

It is possible for these sequences to naturally occur in script expressions, as in the following examples:

```
if (x<!--y) { ... }
if ( player<script ) { ... }
```

In such cases the characters cannot be escaped, but the expressions can be rewritten so that the sequences don't occur, as in:

```
if (x < !--y) { ... }
if (!--y > x) { ... }
if (!(--y) > x) { ... }
if (player < script) { ... }
if (script > player) { ... }
```

Doing this also avoids a different pitfall as well: for related historical reasons, the string `"<!--"` in [classic scripts](#) is actually treated as a line comment start, just like `"//"`.

4.12.1.4. Inline documentation for external scripts

If a `<script>` element's `src` attribute is specified, then the contents of the `<script>` element, if any, must be such that the value of the `text` IDL attribute, which is derived from the element's contents, matches the documentation production in the following ABNF, the character set for which is Unicode. [\[ABNF\]](#)

```
documentation = *( *( space / tab / comment ) [ line-comment ] newline )
comment       = slash star *( not-star / star not-slash ) 1*star slash
line-comment  = slash slash *not-newline

; characters
tab           = %x0009 ; U+0009 CHARACTER TABULATION (tab)
newline      = %x000A ; U+000A LINE FEED (LF)
space        = %x0020 ; U+0020 SPACE
star         = %x002A ; U+002A ASTERISK (*)
slash        = %x002F ; U+002F SOLIDUS (/)
not-newline  = %x0000-0009 / %x000B-10FFFF
              ; a Unicode character other than U+000A LINE FEED (LF)
not-star     = %x0000-0029 / %x002B-10FFFF
              ; a Unicode character other than U+002A ASTERISK (*)
not-slash    = %x0000-002E / %x0030-10FFFF
              ; a Unicode character other than U+002F SOLIDUS (/)
```

NOTE:

This corresponds to putting the contents of the element in JavaScript comments.

NOTE:

This requirement is in addition to the earlier restrictions on the syntax of contents of `<script>` elements.

EXAMPLE 555

This allows authors to include documentation, such as license information or API information, inside their documents while still referring to external script files. The syntax is constrained so that authors don't accidentally include what looks like valid script while also providing a `src` attribute.

```
<script src="cool-effects.js">
  // create new instances using:
  //   var e = new Effect();
  // start the effect using .play, stop using .stop:
  //   e.play();
  //   e.stop();
</script>
```

4.12.1.5. Interaction of `<script>` elements and XSLT

This section is non-normative.

This specification does not define how XSLT interacts with the `<script>` element. However, in the absence of another specification actually defining this, here are some guidelines for implementors, based on existing implementations:

- When an XSLT transformation program is triggered by an `<?xml-stylesheet?>` processing instruction and the browser implements a direct-to-DOM transformation, `<script>` elements created by the XSLT processor need to be marked `"parser-inserted"` and run in document order (modulo scripts marked `defer` or `async`), `immediately`, as the transformation is occurring.
- The `XSLTProcessor.transformToDocument()` method adds elements to a `Document` that is not in a `browsing context`, and, accordingly, any `<script>` elements they create need to have their `"already started"` flag set in the `prepare a script` algorithm and never get executed (`scripting is disabled`). Such `<script>` elements still need to be marked `"parser-inserted"`, though, such that their `async` IDL attribute will return false in the absence of an `async` content attribute.
- The `XSLTProcessor.transformToFragment()` method needs to create a fragment that is equivalent to one built manu-

ally by creating the elements using `document.createElementNS()`. For instance, it needs to create `<script>` elements that aren't "parser-inserted" and that don't have their "already started" flag set, so that they will execute when the fragment is [inserted into a document](#).

The main distinction between the first two cases and the last case is that the first two operate on [Documents](#) and the last operates on a fragment.

§ 4.12.2. The `noscript` element

Categories:

[Metadata content](#).

[Flow content](#).

[Phrasing content](#).

Contexts in which this element can be used:

In a `<head>` element of an [HTML document](#), if there are no ancestor `<noscript>` elements.

Where [phrasing content](#) is expected in [HTML documents](#), if there are no ancestor `<noscript>` elements.

Content model:

When [scripting is disabled](#), in a `<head>` element: in any order, zero or more `<link>` elements, zero or more `<style>` elements, and zero or more `<meta>` elements.

When [scripting is disabled](#), not in a `<head>` element: [transparent](#), but there must be no `<noscript>` element descendants.

Otherwise: text that conforms to the requirements given in the prose.

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed [ARIA role attribute](#) values:

None

Allowed [ARIA state and property attributes](#):

[Global aria-* attributes](#)

DOM interface:

Uses [HTMLElement](#).

The `<noscript>` element [represents](#) nothing if [scripting is enabled](#), and [represents](#) its children if [scripting is disabled](#). It is used to present different markup to user agents that support scripting and those that don't support scripting, by affecting how the document is parsed.

When used in [HTML documents](#), the allowed content model is as follows:

In a `<head>` element, if [scripting is disabled](#) for the `<noscript>` element

The `<noscript>` element must contain only `<link>`, `<style>`, and `<meta>` elements.

In a `<head>` element, if [scripting is enabled](#) for the `<noscript>` element

The `<noscript>` element must contain only text, except that invoking the [HTML fragment parsing algorithm](#) with the `<noscript>` element as the [context](#) element and the text contents as the *input* must result in a list of nodes that consists only of `<link>`, `<style>`, and `<meta>` elements that would be conforming if they were children of the `<noscript>` element, and no [parse errors](#).

Outside of `<head>` elements, if [scripting is disabled](#) for the `<noscript>` element

The `<noscript>` element's content model is [transparent](#), with the additional restriction that a `<noscript>` element must not have a `<noscript>` element as an ancestor (that is, `<noscript>` can't be nested).

Outside of `<head>` elements, if [scripting is enabled](#) for the `<noscript>` element

The `<noscript>` element must contain only text, except that the text must be such that running the following algorithm results in a [conforming document](#) with no `<noscript>` elements and no `<script>` elements, and such that no step in the

algorithm throws an exception or causes an [HTML parser](#) to flag a [parse error](#):

1. Remove every [<script>](#) element from the document.
2. Make a list of every [<noscript>](#) element in the document. For every [<noscript>](#) element in that list, perform the following steps:
 1. Let s be the concatenation of all the [Text](#) node children of the [<noscript>](#) element.
 2. Set the outerHTML attribute of the [<noscript>](#) element to the value of s . (This, as a side-effect, causes the [<noscript>](#) element to be removed from the document.) [\[DOM-Parsing\]](#)

NOTE:

All these contortions are required because, for historical reasons, the [<noscript>](#) element is handled differently by the [HTML parser](#) based on whether [scripting was enabled or not](#) when the parser was invoked.

The [<noscript>](#) element must not be used in [XML documents](#).

NOTE:

The [<noscript>](#) element is only effective in [the HTML syntax](#), it has no effect in [the XHTML syntax](#). This is because the way it works is by essentially "turning off" the parser when scripts are enabled, so that the contents of the element are treated as pure text and not as real elements. XML does not define a mechanism by which to do this.

The [<noscript>](#) element has no other requirements. In particular, children of the [<noscript>](#) element are not exempt from [§4.10.22 Form submission](#), scripting, and so forth, even when [scripting is enabled](#) for the element.

EXAMPLE 556

In the following example, a `<noscript>` element is used to provide fallback for a script.

```
<form action="calcSquare.php">
  <p>
    <label for=x>Number</label>:
    <input id="x" name="x" type="number">
  </p>
  <script>
    var x = document.getElementById('x');
    var output = document.createElement('p');
    output.textContent = 'Type a number; it will be squared right then!';
    x.form.appendChild(output);
    x.form.onsubmit = function () { return false; }
    x.oninput = function () {
      var v = x.valueAsNumber;
      output.textContent = v + ' squared is ' + v * v;
    };
  </script>
  <noscript>
    <input type=submit value="Calculate Square">
  </noscript>
</form>
```

When script is disabled, a button appears to do the calculation on the server side. When script is enabled, the value is computed on-the-fly instead.

The `<noscript>` element is a blunt instrument. Sometimes, scripts might be enabled, but for some reason the page's script might fail. For this reason, it's generally better to avoid using `<noscript>`, and to instead design the script to change the page from being a scriptless page to a scripted page on the fly, as in the next example:

```
<form action="calcSquare.php">
  <p>
    <label for=x>Number</label>:
    <input id="x" name="x" type="number">
  </p>
  <input id="submit" type=submit value="Calculate Square">
  <script>
    var x = document.getElementById('x');
    var output = document.createElement('p');
    output.textContent = 'Type a number; it will be squared right then!';
    x.form.appendChild(output);
    x.form.onsubmit = function () { return false; }
    x.oninput = function () {
      var v = x.valueAsNumber;
      output.textContent = v + ' squared is ' + v * v;
    };
    var submit = document.getElementById('submit');
    submit.parentNode.removeChild(submit);
  </script>
</form>
```

The above technique is also useful in XHTML, since `<noscript>` is not supported in [the XHTML syntax](#).

§ 4.12.3. The template element

Categories:

[Metadata content.](#)

[Flow content.](#)

[Phrasing content](#).

[Script-supporting element](#).

Contexts in which this element can be used:

Where [metadata content](#) is expected.

Where [phrasing content](#) is expected.

Where [script-supporting elements](#) are expected.

As a child of a [<colgroup>](#) element that doesn't have a `span` attribute.

Content model:

Either: [Metadata content](#).

Or: [Flow content](#).

Or: The content model of `ol` and [](#) elements.

Or: The content model of [<dl>](#) elements.

Or: The content model of [<figure>](#) elements.

Or: The content model of [<ruby>](#) elements.

Or: The content model of [<object>](#) elements.

Or: The content model of video and [<audio>](#) elements.

Or: The content model of [<table>](#) elements.

Or: The content model of [<colgroup>](#) elements.

Or: The content model of [<thead>](#), [<tbody>](#), and [<tfoot>](#) elements.

Or: The content model of [<tr>](#) elements.

Or: The content model of [<fieldset>](#) elements.

Or: The content model of [<select>](#) elements.

Or: The content model of [<details>](#) elements.

Or: The content model of [<menu>](#) elements whose `type` attribute is in the [popup menu](#) state.

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)

Allowed ARIA role attribute values:

None

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)

DOM interface:

```
interface HTMLTemplateElement : HTMLElement {
    readonly attribute DocumentFragment content;
};
```

The [<template>](#) element is used to declare fragments of HTML that can be cloned and inserted in the document by script.

NOTE:

Templates provide a method for declaring inert DOM subtrees and manipulating them to instantiate document fragments with identical contents.

NOTE:

When web pages dynamically alter the contents of their documents (e.g., in response to user interaction or new data arriving from the server), it is common that they require fragments of HTML which may require further modification before use, such as the insertion of values appropriate for the usage context.

NOTE:

The `<template>` element allows for the declaration of document fragments which are unused by the document when loaded, but are parsed as HTML and are available at runtime for use by the web page.

In a rendering, the `<template>` element **represents** nothing.

This definition is non-normative. Implementation requirements are given below this definition.

`template` . content

Returns the contents of the `<template>`, which are stored in a `DocumentFragment` associated with a different `Document` so as to avoid the `<template>` contents interfering with the main `Document`. (For example, this avoids form controls from being submitted, scripts from executing, and so forth.)

Each `<template>` element has an associated `DocumentFragment` object that is its **template contents**. When a `<template>` element is created, the user agent must run the following steps to establish the **template contents**:

1. Let `doc` be the `<template>` element's **node document's appropriate template contents owner document**.
2. Create a `DocumentFragment` object whose **node document** is `doc`.
3. Set the `<template>` element's **template contents** to the newly created `DocumentFragment` object.

A `Document` `doc`'s **appropriate template contents owner document** is the `Document` returned by the following algorithm:

1. If `doc` is not a `Document` created by this algorithm, run these substeps:
 1. If `doc` does not yet have an **associated inert template document** then run these substeps:
 1. Let `new doc` be a new `Document` (that does not have a **browsing context**). This is "a `Document` created by this algorithm" for the purposes of the step above.
 2. If `doc` is an **HTML document**, mark `new doc` as an **HTML document** also.
 3. Let `doc`'s **associated inert template document** be `new doc`.
 2. Set `doc` to `doc`'s **associated inert template document**.

NOTE:

Each `Document` not created by this algorithm thus gets a single `Document` to act as its proxy for owning the **template contents** of all its `<template>` elements, so that they aren't in a **browsing context** and thus remain inert (e.g., scripts do not run). Meanwhile, `<template>` elements inside `Document` objects that are created by this algorithm just reuse the same `Document` owner for their contents.

2. Return `doc`.

The **adopting steps** (with `node` and `oldDocument` as parameters) for `<template>` elements are the following:

1. Let `doc` be `node`'s **node document's appropriate template contents owner document**.

NOTE:

`node`'s **node document** is the `Document` object that `node` was just adopted into.

2. **Adopt** `node`'s **template contents** (a `DocumentFragment` object) into `doc`.

The **content** IDL attribute must return the `<template>` element's **template contents**.



The **cloning steps** for a template element `node` being cloned to a copy `copy` must run the following steps:

1. If the **clone children flag** is not set in the calling **clone** algorithm, abort these steps.
2. Let `copied contents` be the result of **cloning** all the children of `node`'s **template contents**, with `document` set to `copy`'s

[template contents](#)'s [node document](#), and with the `clone children flag` set.

3. Append `copied contents` to `copy`'s [template contents](#).

EXAMPLE 557

In this example, a script populates a table four-column with data from a data structure, using a `template` to provide the element structure instead of manually generating the structure from markup.

```
<!DOCTYPE html>
<title>Cat data</title>
<script>
  // Data is hard-coded here, but could come from the server
  var data = [
    { name: 'Pillar', color: 'Ticked Tabby', sex: 'Female (neutered)', legs: 3 },
    { name: 'Hedral', color: 'Tuxedo', sex: 'Male (neutered)', legs: 4 },
  ];
</script>
<table>
  <thead>
    <tr>
      <th>Name <th>Color <th>Sex <th>Legs
    </thead>
    <tbody>
      <template id="row">
        <tr><td><td><td><td>
      </template>
    </tbody>
  </table>
<script>
  var template = document.querySelector('#row');
  for (var i = 0; i < data.length; i += 1) {
    var cat = data[i];
    var clone = template.content.cloneNode(true);
    var cells = clone.querySelectorAll('td');
    cells[0].textContent = cat.name;
    cells[1].textContent = cat.color;
    cells[2].textContent = cat.sex;
    cells[3].textContent = cat.legs;
    template.parentNode.appendChild(clone);
  }
</script>
```

This example uses `cloneNode()` on the `template`'s contents; it could equivalently have used `document.importNode()`, which does the same thing. The only difference between these two APIs is when the [node document](#) is updated: with `cloneNode()` it is updated when the nodes are appended with `appendChild()`, with `document.importNode()` it is updated when the nodes are cloned.

4.12.3.1. Interaction of `<template>` elements with XSLT and XPath

This section is non-normative.

This specification does not define how XSLT and XPath interact with the `template` element. However, in the absence of another specification actually defining this, here are some guidelines for implementors, which are intended to be consistent with other processing described in this specification:

- An XSLT processor based on an XML parser that acts as described in this specification needs to act as if `<template>` elements contain as descendants their [template contents](#) for the purposes of the transform.
- An XSLT processor that outputs a DOM needs to ensure that nodes that would go into a `<template>` element are instead placed into the element's [template contents](#).
- XPath evaluation using the XPath DOM API when applied to a Document parsed using the [HTML parser](#) or the [XML parser](#) described in this specification needs to ignore [template contents](#).

§ 4.12.4. The canvas element

Categories:

[Flow content.](#)
[Phrasing content.](#)
[Embedded content.](#)
[Palpable content.](#)

Contexts in which this element can be used:

Where [embedded content](#) is expected.

Content model:

[Transparent.](#)

Tag omission in text/html:

Neither tag is omissible

Content attributes:

[Global attributes](#)
width - Horizontal dimension
height - Vertical dimension

Allowed ARIA role attribute values:

[Any role value.](#)

Allowed ARIA state and property attributes:

[Global aria-* attributes](#)
Any aria-* attributes [applicable to the allowed roles.](#)

DOM interface:

```
typedef (CanvasRenderingContext2D or WebGLRenderingContext) RenderingContext;

interface HTMLCanvasElement : HTMLElement {
    attribute unsigned long width;
    attribute unsigned long height;

    RenderingContext? getContext(DOMString contextId, any... arguments);
    boolean probablySupportsContext(DOMString contextId, any... arguments);

    DOMString toDataURL(optional DOMString type, any... arguments);
    void toBlob(BlobCallback _callback, optional DOMString type, any... arguments);
};

callback BlobCallback = void (Blob? blob);
```

The [<canvas>](#) element provides scripts with a resolution-dependent bitmap canvas, which can be used for rendering graphs, game graphics, art, or other visual images on the fly.

Authors should not use the [<canvas>](#) element in a document when a more suitable element is available. For example, it is inappropriate to use a [<canvas>](#) element to render a page heading: if the desired presentation of the heading is graphically intense, it should be marked up using appropriate elements (typically h1) and then styled using CSS and supporting technologies such as Web Components.

When authors use the [<canvas>](#) element, they must also provide content that, when presented to the user, conveys essentially the same function or purpose as the [<canvas>](#)'s bitmap. This content may be placed as content of the [<canvas>](#) element. The contents of the [<canvas>](#) element, if any, are the element's [fallback content](#).



In interactive visual media, if [scripting is enabled](#) for the [<canvas>](#) element, and if support for [<canvas>](#) elements has been enabled, the [<canvas>](#) element [represents embedded content](#) consisting of a dynamically created image, the element's bit-

map.

In non-interactive, static, visual media, if the [<canvas>](#) element has been previously associated with a rendering context (e.g., if the page was viewed in an interactive visual medium and is now being printed, or if some script that ran during the page layout process painted on the element), then the [<canvas>](#) element [represents embedded content](#) with the element's current bitmap and size. Otherwise, the element represents its [fallback content](#) instead.

In non-visual media, and in visual media if [scripting is disabled](#) for the [<canvas>](#) element or if support for [<canvas>](#) elements has been disabled, the [<canvas>](#) element [represents](#) its [fallback content](#) instead.

When a [<canvas>](#) element [represents embedded content](#), the user can still focus descendants of the [<canvas>](#) element (in the [fallback content](#)). When an element is [focused](#), it is the target of keyboard interaction events (even though the element itself is not visible). This allows authors to make an interactive canvas keyboard-accessible: authors should have a one-to-one mapping of interactive regions to *focusable areas* in the [fallback content](#). (Focus has no effect on mouse interaction events.) [\[UIEVENTS\]](#)

An element whose nearest [<canvas>](#) element ancestor is [being rendered](#) and [represents embedded content](#) is an element that is [being used as relevant canvas fallback content](#).



The [<canvas>](#) element has two attributes to control the size of the element's bitmap: **width** and **height**. These attributes, when specified, must have values that are [valid non-negative integers](#). The [rules for parsing non-negative integers](#) must be used to obtain their numeric values. If an attribute is missing, or if parsing its value returns an error, then the default value must be used instead. The width attribute defaults to 300, and the height attribute defaults to 150.

The [intrinsic dimensions](#) of the [<canvas>](#) element when it [represents embedded content](#) are equal to the dimensions of the element's bitmap.

The user agent must use a square pixel density consisting of one pixel of image data per coordinate space unit for the bitmaps of a canvas and its rendering contexts.

NOTE:

A [<canvas>](#) element can be sized arbitrarily by a style sheet, its bitmap is then subject to the ['object-fit'](#) CSS property. [\[CSS3-IMAGES\]](#)



The bitmaps of [<canvas>](#) elements, the bitmaps of `ImageBitmap` objects, as well as some of the bitmaps of rendering contexts, such as those described in the section on the `CanvasRenderingContext2D` object below, have an **origin-clean** flag, which can be set to true or false. Initially, when the [<canvas>](#) element or `ImageBitmap` object is created, its bitmap's [origin-clean](#) flag must be set to true.

A canvas bitmap can also have a hit region list, as described in the `CanvasRenderingContext2D` section below.

A [<canvas>](#) element can have a rendering context bound to it. Initially, it does not have a bound rendering context. To keep track of whether it has a rendering context or not, and what kind of rendering context it is, a canvas also has a **canvas context mode**, which is initially **none** but can be changed to either **2d**, **webgl** by algorithms defined in this specification.

When its [canvas context mode](#) is **none**, a [<canvas>](#) element has no rendering context, and its bitmap must be fully transparent black with an [intrinsic width](#) equal to the numeric value of the element's width attribute and an [intrinsic height](#) equal to the numeric value of the element's height attribute, those values being interpreted in CSS pixels, and being updated as the attributes are set, changed, or removed.

When a [<canvas>](#) element represents [embedded content](#), it [provides a paint source](#) whose width is the element's [intrinsic width](#), whose height is the element's [intrinsic height](#), and whose appearance is the element's bitmap.

Whenever the width and height content attributes are set, removed, changed, or redundantly set to the value they already have, if the [canvas context mode](#) is **2d**, the user agent must set bitmap dimensions to the numeric values of the width and height content attributes.

The **width** and **height** IDL attributes must [reflect](#) the respective content attributes of the same name, with the same defaults.



This definition is non-normative. Implementation requirements are given below this definition.

context = `canvas . getContext(contextId [, ... ] )`

Returns an object that exposes an API for drawing on the canvas. The first argument specifies the desired API, either "2d" or "webgl". Subsequent arguments are handled by that API.

The list of defined contexts is given on the [WHATWG Wiki CanvasContexts page](#). [\[WHATWGWIKI\]](#)

Example contexts are the "2d" [\[CANVAS-2D\]](#) and the "webgl" context [\[WEBGL\]](#).

Returns null if the given context ID is not supported or if the canvas has already been initialized with some other (incompatible) context type (e.g., trying to get a "2d" context after getting a "webgl" context).

supported = `canvas . probablySupportsContext(contextId [, ... ] )`

Returns false if calling `getContext()` with the same arguments would definitely return null, and true otherwise.

This return value is not a guarantee that `getContext()` will or will not return an object, as conditions (e.g., availability of system resources) can vary over time.

The `getContext(contextId, arguments...)` method of the `<canvas>` element, when invoked, must run the steps in the cell of the following table whose column header describes the `<canvas>` element's [canvas context mode](#) and whose row header describes the method's first argument.

getContext() invocation steps

	none	2d	webgl
"2d"	Set the <code><canvas></code> element's context mode to 2d , obtain a <code>CanvasRenderingContext2D</code> object as defined in the HTML Canvas 2D Context specification [CANVAS-2D] , set the obtained <code>CanvasRenderingContext2D</code> object's context mode to 2d , and return the <code>CanvasRenderingContext2D</code> object	Return the same object as was return the last time the method was invoked with this same first argument.	Return null.
"webgl", if the user agent supports the WebGL feature in its current configuration	Follow the instructions given in the WebGL specification's <i>Context Creation</i> section to obtain either a <code>WebGLRenderingContext</code> or null; if the returned value is null, then return null and abort these steps, otherwise, set the <code><canvas></code> element's context mode to webgl , set the new <code>WebGLRenderingContext</code> object's context mode to webgl , and return the <code>WebGLRenderingContext</code> object‡ [WEBGL]	Return null.	Return the same object as was return the last time the method was invoked with this same first argument.
A vendor-specific extension*	Behave as defined for the extension.	Behave as defined for the extension.	Behave as defined for the extension.
An unsupported value†	Return null.	Return null.	Return null.

* Vendors may define experimental contexts using the syntax `vendorname - context`, for example, `moz-3d`.

† For example, the "webgl" value in the case of a user agent having exhausted the graphics hardware's abilities and having no software fallback implementation.

‡ The second (and subsequent) argument(s) to the method, if any, are ignored in all cases except this one. See the WebGL specification for details.



⚠Warning! There is no known native implementation of the **probablySupportsContext()** method. Therefore this feature should not be relied upon.

The `probablySupportsContext(contextId, arguments...)` method of the `<canvas>` element, when invoked, must return false if calling `getContext()` on the same object and with the same arguments would definitely return null at this time, and true otherwise.



This definition is non-normative. Implementation requirements are given below this definition.

`url = canvas.toDataURL([type, ...])`

Returns a [data: URL](#) for the image in the canvas.

The first argument, if provided, controls the type of the image to be returned (e.g., PNG or JPEG). The default is `image/png`; that type is also used if the given type isn't supported. The other arguments are specific to the type, and control the way that the image is generated, as given in the table below.

When trying to use types other than `"image/png"`, authors can check if the image was really returned in the requested format by checking to see if the returned string starts with one of the exact strings `"data:image/png,"` or `"data:image/png;"`. If it does, the image is PNG, and thus the requested type was not supported. (The one exception to this is if the canvas has either no height or no width, in which case the result might simply be `"data:,"`.)

`canvas.toBlob(callback[, type, ...])`

Creates a `Blob` object representing a file containing the image in the canvas, and invokes a callback with a handle to that object.

The second argument, if provided, controls the type of the image to be returned (e.g., PNG or JPEG). The default is `image/png`; that type is also used if the given type isn't supported. The other arguments are specific to the type, and control the way that the image is generated, as given in the table below.

The **`toDataURL()`** method must run the following steps:

1. If the `<canvas>` element's bitmap's [origin-clean](#) flag is set to false, throw a **"SecurityError" DOMException** and abort these steps.
2. If the `<canvas>` element's bitmap has no pixels (i.e., either its horizontal dimension or its vertical dimension is zero) then return the string `"data:,"` and abort these steps. (This is the shortest [data: URL](#); it represents the empty string in a `text/plain` resource.)
3. Let *file* be [a serialization of the canvas element's bitmap as a file](#), using the method's arguments (if any) as the *arguments*.
4. Return a [data: URL](#) representing *file*. [\[RFC2397\]](#)

The **`toBlob()`** method must run the following steps:

1. If the `<canvas>` element's bitmap's [origin-clean](#) flag is set to false, throw a **"SecurityError" DOMException** and abort these steps.
2. Let *callback* be the first argument.
3. Let *arguments* be the second and subsequent arguments to the method, if any.

4. If the `<canvas>` element's bitmap has no pixels (i.e., either its horizontal dimension or its vertical dimension is zero) then let `result` be null.

Otherwise, let `result` be a Blob object representing a **serialization of the canvas element's bitmap as a file**, using *arguments*. [\[FILEAPI\]](#)

5. Return, but continue running these steps **in parallel**.
6. **Queue a task** to invoke the BlobCallback `callback` with `result` as its argument. The **task source** for this task is the **canvas blob serialization task source**.

4.12.4.1. Color spaces and color correction

The canvas APIs must perform color correction at only two points: when rendering images with their own gamma correction and color space information onto a bitmap, to convert the image to the color space used by the bitmaps (e.g., using the 2D Context's `drawImage()` method with an `HTMLImageElement` object), and when rendering the actual canvas bitmap to the output device.

NOTE:

Thus, in the 2D context, colors used to draw shapes onto the canvas will exactly match colors obtained through the `getImageData()` method.

The `toDataURL()` method must not include color space information in the resources they return. Where the output format allows it, the color of pixels in resources created by `toDataURL()` must match those returned by the `getImageData()` method.

In user agents that support CSS, the color space used by a `<canvas>` element must match the color space used for processing any colors for that element in CSS.

The gamma correction and color space information of images must be handled in such a way that an image rendered directly using an `<img>` element would use the same colors as one painted on a `<canvas>` element that is then itself rendered.

Furthermore, the rendering of images that have no color correction information (such as those returned by the `toDataURL()` method) must be rendered with no color correction.

NOTE:

Thus, in the 2D context, calling the `drawImage()` method to render the output of the `toDataURL()` method to the canvas, given the appropriate dimensions, has no visible effect.

4.12.4.2. Serializing bitmaps to a file

When a user agent is to create a **serialization of the bitmap as a file**, optionally with some given *arguments*, and optionally with a *native* flag set, it must create an image file in the format given by the first value of *arguments*, or, if there are no *arguments*, in the PNG format. [\[PNG\]](#)

If the *native* flag is set, or if the bitmap has one pixel per coordinate space unit, then the image file must have the same pixel data (before compression, if applicable) as the bitmap, and if the file format used supports encoding resolution metadata, the resolution of that bitmap (device pixels per coordinate space units being interpreted as image pixels per CSS pixel) must be given as well.

Otherwise, the image file's pixel data must be the bitmap's pixel data scaled to one image pixel per coordinate space unit, and if the file format used supports encoding resolution metadata, the resolution must be given as 96dpi (one image pixel per CSS pixel).

If *arguments* is not empty, the first value must be interpreted as a [MIME type](#) giving the format to use. If the type has any parameters, it must be treated as not supported.

EXAMPLE 558

For example, the value "image/png" would mean to generate a PNG image, the value "image/jpeg" would mean to generate a JPEG image, and the value "image/svg+xml" would mean to generate an SVG image (which would require that the user agent track how the bitmap was generated, an unlikely, though potentially awesome, feature).

User agents must support PNG ("image/png"). User agents may support other types. If the user agent does not support the requested type, it must create the file using the PNG format. [\[PNG\]](#)

User agents must [convert the provided type to ASCII lowercase](#) before establishing if they support that type.

For image types that do not support an alpha channel, the serialized image must be the bitmap image composited onto a solid black background using the source-over operator.

If the first argument in *arguments* gives a type corresponding to one of the types given in the first column of the following table, and the user agent supports that type, then the subsequent arguments, if any, must be treated as described in the second cell of that row.

Arguments for serialization methods

Type	Other arguments	Reference
image/jpeg	The second argument, if it is a number in the range 0.0 to 1.0 inclusive, must be treated as the desired quality level. If it is not a number or is outside that range, the user agent must use its default value, as if the argument had been omitted.	[JPEG]

For the purposes of these rules, an argument is considered to be a number if it is converted to an IDL double value by the rules for handling arguments of type any in the Web IDL specification. [\[WEBIDL\]](#)

Other arguments must be ignored and must not cause the user agent to throw an exception. A future version of this specification will probably define other parameters to be passed to these methods to allow authors to more carefully control compression settings, image metadata, etc.

§ 4.12.4.3. Security with [<canvas>](#) elements

This section is non-normative.

Information leakage can occur if scripts from one [origin](#) can access information (e.g., read pixels) from images from another origin (one that isn't the same).

To mitigate this, bitmaps used with [<canvas>](#) elements and `ImageBitmap` objects are defined to have a flag indicating whether they are [origin-clean](#). All bitmaps start with their [origin-clean](#) set to true. The flag is set to false when cross-origin images or fonts are used.

The `toDataURL()`, `toBlob()`, and `getImageData()` methods check the flag and will throw a "[SecurityError](#)" [DOMException](#) rather than leak cross-origin data.

The value of the [origin-clean](#) flag is propagated from a source [<canvas>](#) element's bitmap to a new `ImageBitmap` object by `createImageBitmap()`. Conversely, a destination [<canvas>](#) element's bitmap will have its [origin-clean](#) flags set to false by `drawImage` if the source image is an `ImageBitmap` object whose bitmap has its [origin-clean](#) flag set to false.

The flag can be reset in certain situations; for example, when a `CanvasRenderingContext2D` is bound to a new [<canvas>](#), the bitmap is cleared and its flag reset.

§ 4.13. Common idioms without dedicated elements

§ 4.13.1. Subheadings, subtitles, alternative titles and taglines

HTML does not have a dedicated mechanism for marking up subheadings, alternative titles or taglines. Here are the suggested alternatives. `h1`–`h6` elements must not be used to markup subheadings, subtitles, alternative titles and taglines unless intended to be the heading for a new section or subsection.

EXAMPLE 559

In the following example the title and subtitles of a web page are grouped using a `<header>` element. As the author does not want the subtitles to be included the table of contents and they are not intended to signify the start of a new section, they are marked up using `<p>` elements. A sample CSS styled rendering of the title and subtitles is provided below the code example.

```
<header>
  <h1>HTML 5.1 Nightly</h1>
  <p>A vocabulary and associated APIs for HTML and XHTML</p>
  <p>Editor's Draft 9 May 2013</p>
</header>
```

HTML 5.1 Nightly

A vocabulary and associated APIs for HTML and XHTML

Editor's Draft 9 May 2013

EXAMPLE 560

In the following example the subtitle of a book is on the same line as the title separated by a colon. A sample CSS styled rendering of the title and subtitle is provided below the code example.

```
<h1>The Lord of the Rings: The Two Towers</h1>
```

The Lord of the Rings: The Two Towers

EXAMPLE 561

In the following example part of an album title is included in a `<span>` element, allowing it to be styled differently from the rest of the title. A `<br>` element is used to place the album title on a new line. A sample CSS styled rendering of the heading is provided below the code example.

```
<h1>Ramones <br>
  <span>Hey! Ho! Let's Go</span>
</h1>
```

RAMONES
Hey! Ho! Let's Go

EXAMPLE 562

In the following example the title and tagline for a news article are grouped using a `<header>` element. The title is marked up using a `h2` element and the tagline is in a `<p>` element. A sample CSS styled rendering of the title and tagline is provided below the code example.

```
<header>
  <h2>3D films set for popularity slide </h2>
  <p>First drop in 3D box office projected for this year despite hotly tipped summer
  blockbusters,
  according to Fitch Ratings report</p>
</header>
```

3D films set for popularity slide

First drop in 3D box office projected for this year despite hotly tipped summer blockbusters, according to Fitch Ratings report

EXAMPLE 563

In this last example the title and taglines for a news magazine are grouped using a `<header>` element. The title is marked up using a `h1` element and the taglines are each in a `<p>` element. A sample CSS styled rendering of the title and taglines is provided below the code example.

```
<header>
  <p>Magazine of the Decade</p>
  <h1>THE MONTH</h1>
  <p>The Best of UK and Foreign Media</p>
</header>
```



§ 4.13.2. Bread crumb navigation

This specification does not provide a machine-readable way of describing bread-crumb navigation menus. Authors are encouraged to markup bread-crumb navigation as a list. The `<nav>` element can be used to mark the list containing links as being a navigation block.

EXAMPLE 564

In the following example, the current page can be reached via the path indicated. The path is indicated using the right arrow symbol "→". A text label is provided to give the user context. The links are structured as a list, which provides users with an indication of item number.

```
<nav>
  <h2>You are here:</h2>
  <ul id="navlist">
    <li><a href="/">Main</a> →</li>
    <li><a href="/products/">Products</a> →</li>
    <li><a href="/products/dishwashers/">Dishwashers</a> →</li>
    <li><a>Second hand</a></li>
  </ul>
</nav>
```

The breadcrumb code example could be styled as a horizontal list using CSS:

You are here: [Main](#) → [Products](#) → [Dishwashers](#) → **Second hand**

NOTE:

The use of the right angle bracket symbol ">" to indicate path direction is discouraged as its meaning, in the context used, is not clearly conveyed to all users.

§ 4.13.3. Tag clouds

This specification does not define any markup specifically for marking up lists of keywords that apply to a group of pages (also known as *tag clouds*). In general, authors are encouraged to either mark up such lists using `<ul>` elements with explicit inline counts that are then hidden and turned into a presentational effect using a style sheet, or to use SVG.

EXAMPLE 565

Here, three tags are included in a short tag cloud:

```
<style>
@media screen, print, handheld, tv {
  /* should be ignored by non-visual browsers */
  .tag-cloud > li > span { display: none; }
  .tag-cloud > li { display: inline; }
  .tag-cloud-1 { font-size: 0.7em; }
  .tag-cloud-2 { font-size: 0.9em; }
  .tag-cloud-3 { font-size: 1.1em; }
  .tag-cloud-4 { font-size: 1.3em; }
  .tag-cloud-5 { font-size: 1.5em; }
}
</style>
...
<ul class="tag-cloud">
  <li class="tag-cloud-4"><a title="28 instances" href="/t/apple">apple</a> <span>(popular)
</span>
  <li class="tag-cloud-2"><a title="6 instances" href="/t/kiwi">kiwi</a> <span>(rare)</span>
  <li class="tag-cloud-5"><a title="41 instances" href="/t/pear">pear</a> <span>(very popular)
</span>
</ul>
```

The actual frequency of each tag is given using the `title` attribute. A CSS style sheet is provided to convert the markup into a cloud of differently-sized words, but for user agents that do not support CSS or are not visual, the markup contains annotations like "(popular)" or "(rare)" to categorize the various tags by frequency, thus enabling all users to benefit from the information.

The `<ul>` element is used (rather than `ol`) because the order is not particularly important: while the list is in fact ordered alphabetically, it would convey the same information if ordered by, say, the length of the tag.

The tag `rel-keyword` is *not* used on these `<a>` elements because they do not represent tags that apply to the page itself; they are just part of an index listing the tags themselves.

§ **4.13.4. Conversations**

This specification does not define a specific element for marking up conversations, meeting minutes, chat transcripts, dialogs in screenplays, instant message logs, and other situations where different players take turns in discourse.

Instead, authors are encouraged to mark up conversations using `<p>` elements and punctuation. Authors who need to mark the speaker for styling purposes are encouraged to use `<span>` or `<b>`. Paragraphs with their text wrapped in the `i` element can be used for marking up stage directions.

EXAMPLE 566

This example demonstrates this using an extract from Abbot and Costello's famous sketch, *Who's on first*:

```
<p> Costello: Look, you gotta first baseman?
<p> Abbott: Certainly.
<p> Costello: Who's playing first?
<p> Abbott: That's right.
<p> Costello becomes exasperated.
<p> Costello: When you pay off the first baseman every month, who gets the money?
<p> Abbott: Every dollar of it.
```

EXAMPLE 567

The following extract shows how an IM conversation log could be marked up, using the `<data>` element to provide Unix timestamps for each line. Note that the timestamps are provided in a format that the `<time>` element does not support, so the `<data>` element is used instead (namely, Unix `time_t` timestamps). Had the author wished to mark up the data using one of the date and time formats supported by the `<time>` element, that element could have been used instead of `data`. This could be advantageous as it would allow data analysis tools to detect the timestamps unambiguously, without coordination with the page author.

```
<p> <data value="1319898155">14:22</data> <b>egof</b> I'm not that nerdy, I've only seen 30%
of the star trek episodes
<p> <data value="1319898192">14:23</data> <b>kaj</b> if you know what percentage of the star
trek episodes you have seen, you are inarguably nerdy
<p> <data value="1319898200">14:23</data> <b>egof</b> it's unarguably
<p> <data value="1319898228">14:23</data> <i>* kaj blinks</i>
<p> <data value="1319898260">14:24</data> <b>kaj</b> you are not helping your case
```


EXAMPLE 568

HTML does not have a good way to mark up graphs, so descriptions of interactive conversations from games are more difficult to mark up. This example shows one possible convention using `<dl>` elements to list the possible responses at each point in the conversation. Another option to consider is describing the conversation in the form of a DOT file, and outputting the result as an SVG image to place in the document. [\[DOT\]](#)

```
<p> Next, you meet a fisherman. You can say one of several greetings:
<dl>
  <dt> "Hello there!"
  <dd>
    <p> He responds with "Hello, how may I help you?"; you can respond with:
    <dl>
      <dt> "I would like to buy a fish."
      <dd> <p> He sells you a fish and the conversation finishes.
      <dt> "Can I borrow your boat?"
      <dd>
        <p> He is surprised and asks "What are you offering in return?".
        <dl>
          <dt> "Five gold." (if you have enough)
          <dt> "Ten gold." (if you have enough)
          <dt> "Fifteen gold." (if you have enough)
          <dd> <p> He lends you the boat. The conversation ends.
          <dt> "A fish." (if you have one)
          <dt> "A newspaper." (if you have one)
          <dt> "A pebble." (if you have one)
          <dd> <p> "No thanks", he replies. Your conversation options
            at this point are the same as they were after asking to borrow
            the boat, minus any options you've suggested before.
        </dl>
      </dd>
    </dl>
  </dd>
</dl>
<dt> "Vote for me in the next election!"
<dd> <p> He turns away. The conversation finishes.
<dt> "Sir, are you aware that your fish are running away?"
<dd>
  <p> He looks at you skeptically and says "Fish cannot run, sir".
  <dl>
    <dt> "You got me!"
    <dd> <p> The fisherman sighs and the conversation ends.
    <dt> "Only kidding."
    <dd> <p> "Good one!" he retorts. Your conversation options at this
      point are the same as those following "Hello there!" above.
    <dt> "Oh, then what are they doing?"
    <dd> <p> He looks at his fish, giving you an opportunity to steal
      his boat, which you do. The conversation ends.
  </dl>
</dd>
</dl>
```

EXAMPLE 569

In some games, conversations are simpler: each character merely has a fixed set of lines that they say. In this example, a game FAQ/walkthrough lists some of the known possible responses for each character:

```
<section>
  <h1>Dialog</h1>
  <p><small>Some characters repeat their lines in order each time you interact
  with them, others randomly pick from amongst their lines. Those who respond in
  order have numbered entries in the lists below.</small>
  <h2>The Shopkeeper</h2>
  <ul>
    <li>How may I help you?
    <li>Fresh apples!
    <li>A loaf of bread for madam?
  </ul>
  <h2>The pilot</h2>
  <p>Before the accident:
  <ul>
    <li>I'm about to fly out, sorry!
    <li>Sorry, I'm just waiting for flight clearance and then I'll be off!
  </ul>
  <p>After the accident:
  <ol>
    <li>I'm about to fly out, sorry!
    <li>Ok, I'm not leaving right now, my plane is being cleaned.
    <li>Ok, it's not being cleaned, it needs a minor repair first.
    <li>Ok, ok, stop bothering me! Truth is, I had a crash.
  </ol>
  <h2>Clan Leader</h2>
  <p>During the first clan meeting:
  <ul>
    <li>Hey, have you seen my daughter? I bet she's up to something nefarious again...
    <li>Nice weather we're having today, eh?
    <li>The name is Bailey, Jeff Bailey. How can I help you today?
    <li>A glass of water? Fresh from the well!
  </ul>
  <p>After the earthquake:
  <ol>
    <li>Everyone is safe in the shelter, we just have to put out the fire!
    <li>I'll go and tell the fire brigade, you keep hosing it down!
  </ol>
</section>
```

§ 4.13.5. Footnotes

HTML does not have a dedicated mechanism for marking up footnotes. Here are the suggested alternatives.



For short inline annotations, the `title` attribute could be used.

EXAMPLE 570

In this example, two parts of a dialog are annotated with footnote-like content using the `title` attribute.

```
<p> <b>Customer</b>: Hello! I wish to register a complaint. Hello. Miss?
<p> <b>Shopkeeper</b>: <span title="Colloquial pronunciation of 'What do you'"
>Watcha</span> mean, miss?
<p> <b>Customer</b>: Uh, I'm sorry, I have a cold. I wish to make a complaint.
<p> <b>Shopkeeper</b>: Sorry, <span title="This is, of course, a lie.">we're
closing for lunch</span>.
```

⚠Warning! Relying on the `title` attribute for the visual display of text content is currently discouraged as many user agents do not expose the attribute in an accessible manner as required by this specification (e.g., requiring a pointing device such as a mouse to cause a tooltip to appear, which excludes keyboard-only users and touch-only users, such as anyone with a modern phone or tablet).

NOTE:

If the `title` attribute is used, CSS can be used to draw the reader's attention to the elements with the attribute.

EXAMPLE 571

For example, the following CSS places a dashed line below elements that have a `title` attribute.

```
[title] { border-bottom: thin dashed; }
```



For annotations, the `<a>` element should be used, pointing to an element later in the document. The convention is that the contents of the link be a number in square brackets.

EXAMPLE 572

In this example, a footnote in the dialog links to a paragraph below the dialog. The paragraph then reciprocally links back to the dialog, allowing the user to return to the location of the footnote.

```
<p> Announcer: Number 16: The <i>hand</i>.
<p> Interviewer: Good evening. I have with me in the studio tonight
Mr Norman St John Polevaulter, who for the past few years has been
contradicting people. Mr Polevaulter, why <em>do</em> you
contradict people?
<p> Norman: I don't. <sup><a href="#fn1" id="r1">[1]</a></sup>
<p> Interviewer: You told me you did!
...
<section>
  <p id="fn1"><a href="#r1">[1]</a> This is, naturally, a lie,
  but paradoxically if it were true he could not say so without
  contradicting the interviewer and thus making it false.</p>
</section>
```



For side notes, longer annotations that apply to entire sections of the text rather than just specific words or sentences, the `<aside>` element should be used.

EXAMPLE 573

In this example, a sidebar is given after a dialog, giving it some context.

```
<p> <span class="speaker">Customer</span>: I will not buy this record, it is scratched.
<p> <span class="speaker">Shopkeeper</span>: I'm sorry?
<p> <span class="speaker">Customer</span>: I will not buy this record, it is scratched.
<p> <span class="speaker">Shopkeeper</span>: No no no, this is a tobacconist's.
<aside role="note">
  <p>In 1970, the British Empire lay in ruins, and foreign
    nationalists frequented the streets – many of them Hungarians
    (not the streets – the foreign nationals). Sadly, Alexander
    Yalt has been publishing incompetently-written phrase books.
</aside>
```

NOTE:

In the example above an ARIA role="note", permitted for use on [<aside>](#), has been added to override the default semantics of the [<aside>](#) element, as the use of the element in this context, more closely matches the [‘note’](#) role.



For figures or tables, footnotes can be included in the relevant [figcaption](#) or [<caption>](#) element, or in surrounding prose.

EXAMPLE 574

In this example, a table has cells with footnotes that are given in prose. A [<figure>](#) element is used to give a single legend to the combination of the table and its footnotes.

```
<figure>
  <figcaption>Table 1. Alternative activities for knights.</figcaption>
  <table>
    <tr>
      <th> Activity
      <th> Location
      <th> Cost
    <tr>
      <td> Dance
      <td> Wherever possible
      <td> £0<sup><a href="#fn1">1</a></sup>
    <tr>
      <td> Routines, chorus scenes<sup><a href="#fn2">2</a></sup>
      <td> Undisclosed
      <td> Undisclosed
    <tr>
      <td> Dining<sup><a href="#fn3">3</a></sup>
      <td> Camelot
      <td> Cost of ham, jam, and spam<sup><a href="#fn4">4</a></sup>
    </table>
    <p id="fn1">1. Assumed.</p>
    <p id="fn2">2. Footwork impeccable.</p>
    <p id="fn3">3. Quality described as "well".</p>
    <p id="fn4">4. A lot.</p>
  </figure>
```

§ 4.14. Disabled elements

An element is said to be **actually disabled** if it falls into one of the following categories:

- a [<button>](#) element that is disabled

- an [<input>](#) element that is disabled
- a [<select>](#) element that is disabled
- a [<textarea>](#) element that is disabled
- an [<optgroup>](#) element that has a disabled attribute
- an [<option>](#) element that is disabled
- a [<menuitem>](#) element that has a disabled attribute
- a [<fieldset>](#) element that is a [disabled fieldset](#)

NOTE:

This definition is used to determine what elements [can be focused](#) and which elements match the `:disabled` pseudo-class.

§ 4.15. Matching HTML elements using selectors

§ 4.15.1. Case-sensitivity

The Selectors specification leaves the case-sensitivity of element names, attribute names, and attribute values to be defined by the host language. [\[CSS3-SELECTORS\]](#)

NOTE:

Selectors defines that ID and class selectors, when matched against elements in documents that are in [quirks mode](#), will be matched in an [ASCII case-insensitive](#) manner.

When comparing a CSS element type selector to the names of [html elements](#) in [HTML documents](#), the CSS element type selector must first be [converted to ASCII lowercase](#). The same selector when compared to other elements must be compared according to its original case. In both cases, the comparison is [case-sensitive](#).

When comparing the name part of a CSS attribute selector to the names of namespace-less attributes on [html elements](#) in [HTML documents](#), the name part of the CSS attribute selector must first be [converted to ASCII lowercase](#). The same selector when compared to other attributes must be compared according to its original case. In both cases, the comparison is [case-sensitive](#).

Attribute selectors on an [HTML element](#) in an [HTML document](#) must treat the *values* of attributes with the following names as [ASCII case-insensitive](#), with one exception as noted in [§10 Rendering](#):

- accept
- accept-charset
- align
- alink
- axis
- bgcolor
- charset
- checked
- clear
- codetype
- color
- compact
- declare
- defer
- dir
- direction
- disabled
- enctype
- face
- frame
- hreflang
- http-equiv
- [lang](#)
- language
- link
- media

- method
- multiple
- nohref
- noresize
- noshade
- nowrap
- readonly
- [rel](#)
- rev
- rules
- scope
- scrolling
- selected
- shape
- target
- text
- type (except as specified in [§10 Rendering](#))
- valign
- valuetype
- vlink

All other attribute values and everything else must be treated as entirely [case-sensitive](#) for the purposes of selector matching. This includes:

- IDs and classes in [no-quirks mode](#) and [limited-quirks mode](#)
- the names of elements not in the [HTML namespace](#)
- the names of [html elements](#) in [XML documents](#)
- the names of attributes of elements not in the [HTML namespace](#)
- the names of attributes of [html elements](#) in [XML documents](#)
- the names of attributes that themselves have namespaces

§ 4.15.2. Pseudo-classes

⚠Warning! There is only one known native implementation of `:dir(ltr)` pseudo-class matching (Firefox/Gecko). Therefore this feature should not be relied upon.

There are a number of dynamic selectors that can be used with HTML. This section defines when these selectors match HTML elements. [\[CSS3-SELECTORS\]](#) [\[CSS-UI-3\]](#)

`‘:Link’`

`‘:visited’`

All `<a>` elements that have an [href](#) attribute, all `<area>` elements that have an [href](#) attribute, and all `<link>` elements that have an [href](#) attribute, must match one of `:link` and `:visited`.

Other specifications might apply more specific rules regarding how these elements are to match these pseudo-classes, to mitigate some privacy concerns that apply with straightforward implementations of this requirement.

`‘:active’`

The `:active` pseudo-class is defined to match an element “while an element is *being activated* by the user”.

To determine whether a particular element is *being activated* for the purposes of defining the `:active` pseudo-class only, an HTML user agent must use the first relevant entry in the following list.

If the element has a descendant that is currently matching the `:active` pseudo-class

The element is *being activated*.

If the element is the [labeled control](#) of a `<label>` element that is currently matching `‘:active’`

The element is *being activated*.

If the element is a `<button>` element

If the element is an `<input>` element whose type attribute is in the [submit button](#), [image button](#), [Reset Button](#), or [Button](#) state

The element is *being activated* if it is [in a formal activation state](#) and it is not disabled.

EXAMPLE 575

For example, if the user is using a keyboard to push a button element by pressing the space bar, the element would match this pseudo-class in between the time that the element received the `keydown` event and the time the element received the `keyup` event.

If the element is a `<menuitem>` element

The element is *being activated* if it is [in a formal activation state](#) and it does not have a `disabled` attribute.

If the element is an `<a>` element that has an `href` attribute**If the element is an `<area>` element that has an `href` attribute****If the element is a `<link>` element that has an `href` attribute****If the element has its `tabindex` focus flag set**

The element is *being activated* if it is [in a formal activation state](#).

If the element is *being actively pointed at*

The element is *being activated*.

An element is said to be **in a formal activation state** between the time the user begins to indicate an intent to trigger the element's [activation behavior](#) and either the time the user stops indicating an intent to trigger the element's [activation behavior](#), or the time the element's [activation behavior](#) has finished running, whichever comes first.

An element is said to be **being actively pointed at** while the user indicates the element using a pointing device while that pointing device is in the "down" state (e.g., for a mouse, between the time the mouse button is pressed and the time it is depressed; for a remote control on a television, the time during which the remote control is pointing at the element).

`':hover'`

The `:hover` pseudo-class is defined to match an element "while the user *designates* an element with a pointing device". For the purposes of defining the `:hover` pseudo-class only, an HTML user agent must consider an element as being one that the user *designates* if it is:

- An element that the user indicates using a pointing device.
- An element that has a descendant that the user indicates using a pointing device.
- An element that is the [labeled control](#) of a `<label>` element that is currently matching `':hover'`.

EXAMPLE 576

Consider in particular a fragment such as:

```
<p> <label for=c> <input id=a> </label> <span id=b> <input id=c> </span> </p>
```

If the user designates the element with ID "`<a>`" with their pointing device, then the `<p>` element (and all its ancestors not shown in the snippet above), the `<label>` element, the element with ID "`<a>`", and the element with ID "`c`" will match the `':hover'` pseudo-class. The element with ID "`<a>`" matches it from condition 1, the `label` and `<p>` elements match it because of condition 2 (one of their descendants is designated), and the element with ID "`c`" matches it through condition 3 (its `<label>` element matches `':hover'`). However, the element with ID "`b`" does *not* match `':hover'`: its descendant is not designated, even though it matches `':hover'`.

`':focus'`

For the purposes of the CSS `':focus'` pseudo-class, an **element has the focus** when its [top-level browsing context](#) has the system focus, it is not itself a [browsing context container](#), and it is one of the elements listed in the [focus chain](#) of the [currently focused area of the top-level browsing context](#).

`':enabled'`

The `:enabled` pseudo-class must match any element that is one of the following:

- a `<button>` element that is not disabled
- an `<input>` element that is not disabled
- a `<select>` element that is not disabled

- a [<textarea>](#) element that is not disabled
- an [<optgroup>](#) element that does not have a `disabled` attribute
- an [<option>](#) element that is not disabled
- a [<menuitem>](#) element that does not have a `disabled` attribute
- a [<fieldset>](#) element that is not a [disabled fieldset](#)

`:disabled`

The `:disabled` pseudo-class must match any element that is [actually disabled](#).

`:checked`

The `:checked` pseudo-class must match any element falling into one of the following categories:

- [<input>](#) elements whose `type` attribute is in the [Checkbox](#) state and whose [checkedness](#) state is true
- [<input>](#) elements whose `type` attribute is in the [Radio Button](#) state and whose [checkedness](#) state is true
- [<option>](#) elements whose [selectedness](#) is true
- [<menuitem>](#) elements whose `type` attribute is in the [Checkbox](#) state and that have a `checked` attribute
- [<menuitem>](#) elements whose `type` attribute is in the [Radio](#) state and that have a `checked` attribute

`:indeterminate`

The `:indeterminate` pseudo-class must match any element falling into one of the following categories:

- [<input>](#) elements whose `type` attribute is in the [Checkbox](#) state and whose `indeterminate` IDL attribute is set to true
- [<input>](#) elements whose `type` attribute is in the [Radio Button](#) state and whose [radio button group](#) contains no [<input>](#) elements whose [checkedness](#) state is true.
- [<progress>](#) elements with no `value` content attribute

`:default`

The `:default` pseudo-class must match any element falling into one of the following categories:

- [<button>](#) elements that are their form's [default button](#)
- [<input>](#) elements whose `type` attribute is in the [submit button](#) or [image button](#) state, and that are their form's [default button](#)
- [<input>](#) elements to which the `checked` attribute applies and that have a `checked` attribute
- [<option>](#) elements that have a `selected` attribute

`:valid`

The `:valid` pseudo-class must match any element falling into one of the following categories:

- elements that are [candidates for constraint validation](#) and that [satisfy their constraints](#)
- [<form>](#) elements that are not the [form owner](#) of any elements that themselves are [candidates for constraint validation](#) but do not [satisfy their constraints](#)
- [<fieldset>](#) elements that have no descendant elements that themselves are [candidates for constraint validation](#) but do not [satisfy their constraints](#)

`:invalid`

The `:invalid` pseudo-class must match any element falling into one of the following categories:

- elements that are [candidates for constraint validation](#) but that do not [satisfy their constraints](#)
- [<form>](#) elements that are the [form owner](#) of one or more elements that themselves are [candidates for constraint val-](#)

idation but do not [satisfy their constraints](#)

- `<fieldset>` elements that have of one or more descendant elements that themselves are [candidates for constraint validation](#) but do not [satisfy their constraints](#)

`‘:in-range’`

The `:in-range` pseudo-class must match all elements that are [candidates for constraint validation](#), [have range limitations](#), and that are neither [suffering from an underflow](#) nor [suffering from an overflow](#).

`‘:out-of-range’`

The `:out-of-range` pseudo-class must match all elements that are [candidates for constraint validation](#), [have range limitations](#), and that are either [suffering from an underflow](#) or [suffering from an overflow](#).

`‘:required’`

The `:required` pseudo-class must match any element falling into one of the following categories:

- `<input>` elements that are *required*
- `<select>` elements that have a `required` attribute
- `<textarea>` elements that have a `required` attribute

`‘:optional’`

The `:optional` pseudo-class must match any element falling into one of the following categories:

- `<input>` elements to which the `required` attribute applies that are not *required*
- `<select>` elements that do not have a `required` attribute
- `<textarea>` elements that do not have a `required` attribute

`‘:read-only’`

`‘:read-write’`

The `:read-write` pseudo-class must match any element falling into one of the following categories, which for the purposes of Selectors are thus considered *user-alterable*: [\[CSS3-SELECTORS\]](#)

- `<input>` elements to which the `readonly` attribute applies, and that are *mutable* (i.e., that do not have the `readonly` attribute specified and that are not disabled)
- `<textarea>` elements that do not have a `readonly` attribute, and that are not disabled
- elements that are [editing hosts](#) or [editable](#) and are neither `<input>` elements nor `<textarea>` elements

The `:read-only` pseudo-class must match all other [html elements](#).

`‘:dir(ltr)’`

The `:dir(ltr)` pseudo-class must match all elements whose [directionality](#) is `'ltr'`.

`‘:dir rtl)’`

The `:dir rtl)` pseudo-class must match all elements whose [directionality](#) is `'rtl'`.

NOTE:

Another section of this specification defines the target element used with the `:target` pseudo-class.

NOTE:

This specification does not define when an element matches the `:lang()` dynamic pseudo-class, as it is defined in sufficient detail in a language-agnostic fashion in the Selectors specification. [\[CSS3-SELECTORS\]](#)

§ 5. User interaction

§ 5.1. The hidden attribute

All [HTML elements](#) may have the hidden content attribute set. The hidden attribute is a [boolean attribute](#). When specified on an element, it indicates that the element is not yet, or is no longer, directly relevant to the page's current state, or that it is being used to declare content to be reused by other parts of the page as opposed to being directly accessed by the user. User agents should not render elements that have the hidden attribute specified. This requirement may be implemented indirectly through the style layer. For example, an HTML+CSS user agent could implement these requirements using the rules suggested in [§10 Rendering](#).

NOTE:

Because this attribute is typically implemented using CSS, it's also possible to override it using CSS. For instance, a rule that applies 'display: block' to all elements will cancel the effects of the hidden attribute. Authors therefore have to take care when writing their style sheets to make sure that the attribute is still styled as expected.

EXAMPLE 577

In the following skeletal example, the attribute is used to hide the Web game's main screen until the user logs in:

```
<h1>The Example Game</h1>
<section>
  <h2>Login</h2>
  <form>
    ...
    <!-- calls login() once the user's credentials have been checked -->
  </form>
  <script>
    function login() {
      // switch screens
      document.getElementById('login').hidden = true;
      document.getElementById('game').hidden = false;
    }
  </script>
</section>
<section hidden>
  ...
</section>
```

The hidden attribute must not be used to hide content just from one presentation — if something is marked hidden, it is hidden from all presentations, including, for instance, screen readers.

Elements that are not themselves hidden must not [hyperlink](#) to elements that are hidden. The `for` attributes of `label` and `<output>` elements that are not themselves hidden must similarly not refer to elements that are hidden. In both cases, such references would cause user confusion.

Elements and scripts may, however, refer to elements that are hidden in other contexts.

EXAMPLE 578

For example, it would be incorrect to use the [href](#) attribute to link to a section marked with the hidden attribute. If the content is not applicable or relevant, then there is no reason to link to it.

It would be fine, however, to use the ARIA `aria-describedby` attribute to refer to descriptions that are themselves hidden. While hiding the descriptions implies that they are not useful alone, they could be written in such a way that they are useful in the specific context of being referenced from the images that they describe.

Similarly, a `<canvas>` element with the hidden attribute could be used by a scripted graphics engine as an off-screen buffer, and a form control could refer to a hidden `<form>` element using its `form` attribute.

Accessibility APIs are encouraged to provide a way to expose structured content while marking it as hidden in the default view. Such content should not be perceivable to users in the normal document flow in any modality, whether using Assistive Technology (AT) or mainstream User Agents.

When such features are available, User Agents may use them to expose the full semantics of hidden elements to AT when

appropriate, if such content is referenced indirectly by an [ID reference](#) or [valid hash-name reference](#). This allows ATs to access the structure of these hidden elements upon user request, while keeping the content hidden in all presentations of the normal document flow. Authors who wish to prevent user-initiated viewing of a hidden element should not reference the element with such a mechanism.

Because some User Agents have flattened hidden content when exposing such content to AT, authors should not reference hidden content which would lose essential meaning when flattened.

EXAMPLE 579

For example, it would be incorrect to use the [href](#) attribute to link to a section marked with the `hidden` attribute. If the content is not applicable or relevant, then there is no reason to link to it.

It would be fine, however, to use the ARIA `aria-describedby` attribute to refer to descriptions that are themselves hidden. While hiding the descriptions implies that they are not useful alone, they could be written in such a way that they are useful in the specific context of being referenced from the images that they describe.

Similarly, a `<canvas>` element with the `hidden` attribute could be used by a scripted graphics engine as an off-screen buffer, and a form control could refer to a hidden `<form>` element using its `form` attribute.

Elements in a section hidden by the `hidden` attribute are still active, e.g., scripts and form controls in such sections still execute and submit respectively. Only their presentation to the user changes.

The **hidden** IDL attribute must [reflect](#) the content attribute of the same name.

§ 5.2. Inert subtrees

NOTE:

This section **does not** define or create any content attribute named "inert". This section merely defines an abstract concept of [inertness](#).

A node (in particular elements and text nodes) can be marked as **inert**. When a node is [inert](#), then the user agent must act as if the node was absent for the purposes of targeting user interaction events, may ignore the node for the purposes of text search user interfaces (commonly known as "find in page"), and may prevent the user from selecting text in that node. User agents should allow the user to override the restrictions on search and text selection, however.

EXAMPLE 580

For example, consider a page that consists of just a single [inert](#) paragraph positioned in the middle of a `<body>`. If a user moves their pointing device from the body over to the [inert](#) paragraph and clicks on the paragraph, no `mouseover` event would be fired, and the `mousemove` and `click` events would be fired on the `<body>` element rather than the paragraph.

NOTE:

When a node is inert, it generally cannot be focused. Inert nodes that are [commands](#) will also get disabled.

While a [browsing context container](#) is marked as [inert](#), its [nested browsing context's active document](#), and all nodes in that [Document](#), must be marked as [inert](#).

An entire Document can be marked as **blocked by a modal dialog** *subject*. While a Document is so marked, every node that is [in the Document](#), with the exception of the *subject* element and its descendants, must be marked [inert](#). (The elements excepted by this paragraph can additionally be marked [inert](#) through other means; being part of a modal dialog does not "protect" a node from being marked [inert](#).)

Only one element at a time can mark a Document as being [blocked by a modal dialog](#).

§ 5.3. Activation

Certain elements in HTML have an [activation behavior](#), which means that the user can activate them. This triggers a sequence of events dependent on the activation mechanism, and normally culminating in a `click` event, as described below.

The user agent should allow the user to manually trigger elements that have an [activation behavior](#), for instance using keyboard or voice input, or through mouse clicks. When the user triggers an element with a defined [activation behavior](#) in a manner other than clicking it, the default action of the interaction event must be to [run synthetic click activation steps](#) on the element.

Each element has a `click in progress` flag, initially set to false.

When a user agent is to **run synthetic click activation steps** on an element, the user agent must run the following steps:

1. If the element's `click in progress` flag is set to true, then abort these steps.
2. Set the `click in progress` flag on the element to true.
3. [Run pre-click activation steps](#) on the element.
4. [Fire a click event](#) at the element. If the [run synthetic click activation steps](#) algorithm was invoked because the `click()` method was invoked, then the `isTrusted` attribute must be initialized to false.
5. If this click event is not canceled, [run post-click activation steps](#) on the element.

If the event *is* canceled, the user agent must [run canceled activation steps](#) on the element instead.

6. Set the `click in progress` flag on the element to false.

When a pointing device is clicked, the user agent must [run authentic click activation steps](#) instead of firing the click event.

When a user agent is to **run authentic click activation steps** for a given event *event*, it must follow these steps:

1. Let *target* be the element designated by the user (the target of *event*).
2. If *target* is a `<canvas>` element, run the canvas [MouseEvent](#) rerouting steps. If this changes *event*'s target, then let *target* be the new target.
3. Set the `click in progress` flag on *target* to true.
4. Let *e* be the [nearest activatable element](#) of *target* (defined below), if any.
5. If there is an element *e*, [run pre-click activation steps](#) on it.
6. [Dispatch](#) *event* (the required click event) at *target*.

If there is an element *e* and the click event is not canceled, [run post-click activation steps](#) on element *e*.

If there is an element *e* and the event *is* canceled, [run canceled activation steps](#) on element *e*.

7. Set the `click in progress` flag on *target* to false.

NOTE:

The algorithms above don't run for arbitrary synthetic events dispatched by author script. The `click()` method can be used to make the [run synthetic click activation steps](#) algorithm happen programmatically.

NOTE:

Click-focusing behavior (e.g., the focusing of a text field when user clicks in one) typically happens before the click, when the mouse button is first depressed, and is therefore not discussed here.

Given an element *target*, the **nearest activatable element** is the element returned by the following algorithm:

1. If *target* has a defined [activation behavior](#), then return *target* and abort these steps.
2. If *target* has a parent element, then set *target* to that parent element and return to the first step.
3. Otherwise, there is no [nearest activatable element](#).

When a user agent is to **run pre-click activation steps** on an element, it must run the **pre-click activation steps** defined for that element, if any.

When a user agent is to **run canceled activation steps** on an element, it must run the **canceled activation steps** defined for that element, if any.

When a user agent is to **run post-click activation steps** on an element, it must run the **activation behavior** defined for that element, if any. Activation behaviors can refer to the `click` event that was fired by the steps above leading up to this point.

This definition is non-normative. Implementation requirements are given below this definition.

`element.click()`

Acts as if the element was clicked.

The **`click()`** method must run the following steps:

1. If the element is a form control that is disabled, abort these steps.
2. [Run synthetic click activation steps](#) on the element.

§ 5.4. Focus

§ 5.4.1. Introduction

This section is non-normative.

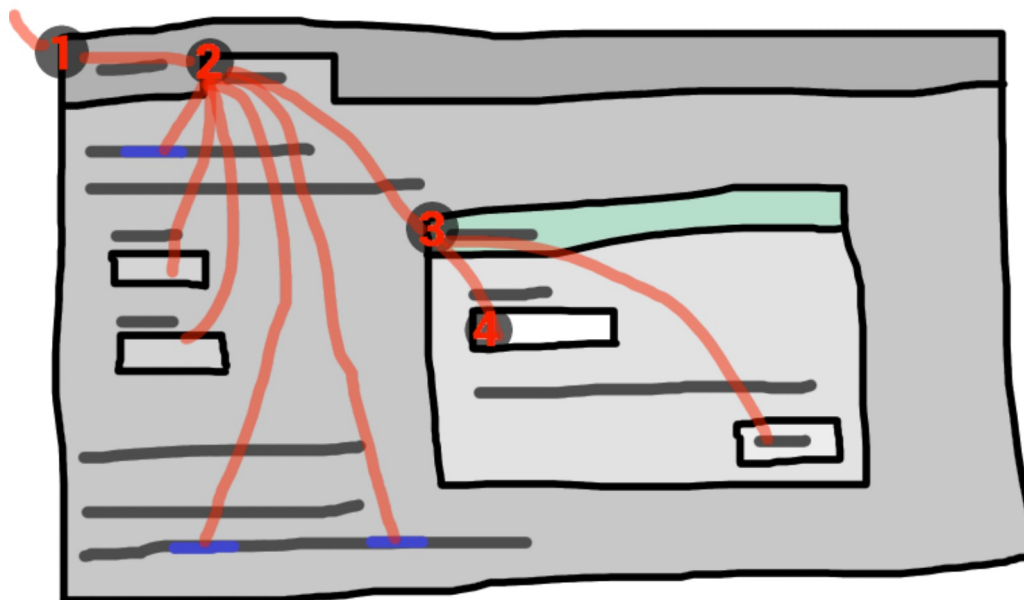
An HTML user interface typically consists of multiple interactive widgets, such as form controls, scrollable regions, links, dialog boxes, browser tabs, and so forth. These widgets form a hierarchy, with some (e.g., browser tabs) containing others (e.g., links, form controls).

When interacting with an interface using a keyboard, key input is channeled from the system, through the hierarchy of interactive widgets, to an active widget, which is said to be [focused](#).

EXAMPLE 581

Consider an HTML application running in a browser tab running in a graphical environment. Suppose this application had a page with some text fields and links, and was currently showing a modal dialog, which itself had a text field and a button.

The hierarchy of focusable widgets, in this scenario, would include the browser window, which would have, amongst its children, the browser tab containing the HTML application. The tab itself would have as its children the various links and text fields, as well as the dialog. The dialog itself would have as its children the text field and the button.



If the widget with [focus](#) in this example was the text field in the dialog box, then key input would be channeled from the graphical system to ① the Web browser, then to ② the tab, then to ③ the dialog, and finally to ④ the text field.

Keyboard *events* are always targeted at this [focused](#) element.

§ 5.4.2. Data model

The term **focusable area** is used to refer to regions of the interface that can become the target of keyboard input. Focusable areas can be elements, parts of elements, or other regions managed by the user agent.

Each [focusable area](#) has a **DOM anchor**, which is a Node object that represents the position of the [focusable area](#) in the DOM. (When the [focusable area](#) is itself a Node, it is its own [DOM anchor](#).) The [DOM anchor](#) is used in some APIs as a substitute for the [focusable area](#) when there is no other DOM object to represent the [focusable area](#).

The following table describes what objects can be [focusable areas](#). The cells in the left column describe objects that can be [focusable areas](#); the cells in the right column describe the [DOM anchors](#) for those elements. (The cells that span both columns are non-normative examples.)

Focusable area	DOM anchor
Examples	
Elements that have their tabindex focus flag set, that are not actually disabled , that are not expressly inert , and that are either being rendered or being used as relevant canvas fallback content .	The element itself.

EXAMPLE 582

[<iframe>](#), [<input type=text>](#), sometimes [](#) (depending on platform conventions).

Focusable area	DOM anchor
Examples	
The shapes of <area> elements in an image map associated with an element that is being rendered and is not expressly inert.	The element.
EXAMPLE 583 In the following example, the <area> element creates two shapes, one on each image. The DOM anchor of the first shape is the first element, and the DOM anchor of the second shape is the second element. <pre><map id=wallmap><area alt="Enter Door" coords="10,10,100,200" href="door.html"></map> </pre>	
The user-agent provided subwidgets of elements that are being rendered and are not actually disabled or expressly inert.	The element for which the focusable area is a subwidget.
EXAMPLE 584 The controls in the user interface that is exposed to the user for a <video> element, the up and down buttons in a spin-control version of <code><input type=number></code>, the part of a <details> element's rendering that enabled the element to be opened or closed using keyboard input.	
The scrollable regions of elements that are being rendered and are not expressly inert.	The element for which the box that the scrollable region scrolls was created.
EXAMPLE 585 The CSS ‘overflow’ property's ‘scroll’ value typically creates a scrollable region.	
The viewport of a Document that is in a browsing context and is not inert.	The Document for which the viewport was created.
EXAMPLE 586 The contents of an <iframe>.	
Any other element or part of an element, especially to aid with accessibility or to better match platform conventions.	The element.
EXAMPLE 587 A user agent could make all list item bullets focusable, so that a user can more easily navigate lists.	
EXAMPLE 588 Similarly, a user agent could make all elements with <code>title</code> attributes focusable, so that their advisory information can be accessed.	

NOTE:

A [browsing context container](#) (e.g., an [<iframe>](#)) is a [focusable area](#), but key events routed to a [browsing context container](#) get immediately routed to the [nested browsing context's active document](#). Similarly, in sequential focus navigation a [browsing context container](#) essentially acts merely as a placeholder for its [nested browsing context's active document](#).

Each [focusable area](#) belongs to a **control group**. Each [control group](#) has an **owner**. [Control group owners](#) are **control group owner objects**. The following are [control group owner objects](#):

- [Document](#) object that have [browsing contexts](#).

Each [control group owner object](#) owns one [control group](#) (though that group might be empty).

If the [DOM anchor](#) of a [focusable area](#) is a [control group owner object](#), then that [focusable area](#) belongs to that [control group owner object's control group](#). Otherwise, the [focusable area](#) belongs to its [DOM anchor's](#) nearest ancestor [control group owner object](#).

EXAMPLE 589

Thus, a [viewport](#) always belongs to the [control group](#) of the Document for which the [viewport](#) was created, an `<input>` control belongs to the [control group](#) of its nearest ancestor [Document](#), and an image map's shapes belong to the nearest ancestor Document of the `<img>` elements (not the `<area>` elements — this means one `<area>` element might create multiple shapes in different [control groups](#)).

An element is **expressly inert** if it is [inert](#) but it is not a [control group owner object](#) and its nearest ancestor [control group owner object](#) is not [inert](#).

One [focusable area](#) in each non-empty [control group](#) is designated the **focused area of the control group**. Which control is so designated changes over time, based on algorithms in this specification. If a [control group](#) is empty, it has no [focused area](#).



[Focusable areas](#) in [control groups](#) are ordered relative to the [tree order](#) of their [DOM anchors](#). [Focusable areas](#) with the same [DOM anchor](#) in a [control group](#) are ordered relative to their CSS box's relative positions in a pre-order, depth-first traversal of the box tree. [CSS-2015]



The **currently focused area of a top-level browsing context** at any particular time is the [focusable area](#) returned by this algorithm:

1. Let *candidate* be the Document of the [top-level browsing context](#).
2. If *candidate* has a non-empty [control group](#), and the designated [focused area of the control group](#) is a [browsing context container](#), then let *candidate* be the [active document](#) of that [browsing context container's nested browsing context](#), and redo this step.

Otherwise, if *candidate* has a non-empty [control group](#), let *candidate* be the designated [focused area of the control group](#).

3. Return *candidate*.

An element that is the [DOM anchor](#) of a [focusable area](#) is said to **gain focus** when that [focusable area](#) becomes the [currently focused area of a top-level browsing context](#). When an element is the [DOM anchor](#) of a [focusable area](#) of the [currently focused area of a top-level browsing context](#), it is **focused**.

The **focus chain** of a [focusable area](#) or [control group owner object](#) *subject* is the ordered list constructed as follows:

1. Let *current object* be *subject*.
2. Let *output* be an empty list.
3. *Loop*: Append *current object* to *output*.
4. If *current object* is an `<area>` element's shape, append that `<area>` element to *output*.

Otherwise, if *current object* is a [focusable area](#) whose [DOM anchor](#) is an element that is not *current object* itself, append that [DOM anchor](#) element to *output*.

5. If *current object* is a [focusable area](#), let *current object* be that [focusable area's control group's owner](#), and return to the step labeled *loop*.

Otherwise, if *current object* is a Document in a [nested browsing context](#), let *current object* be its [browsing context container](#), and return to the step labeled *loop*.

6. Return *output*.

NOTE:

The chain starts with *subject* and (if *subject* is or can be the [currently focused area of a top-level browsing context](#)) continues up the focus hierarchy up to the Document of the [top-level browsing context](#).

§ 5.4.3. The `tabindex` attribute

The **`tabindex`** content attribute allows authors to indicate that an element is supposed to be [focusable](#), and whether it is supposed to be reachable using [sequential focus navigation](#) and, if so, what is to be the relative order of the element for the purposes of sequential focus navigation. The name "tab index" comes from the common use of the "tab" key to navigate through the focusable elements. The term "tabbing" refers to moving forward through the focusable elements that can be reached using sequential focus navigation.

When the attribute is omitted, the user agent applies defaults. (There is no way to make an element that is [being rendered](#) be not focusable at all without [disabling](#) it or making it [inert](#).)

The `tabindex` attribute, if specified, must have a value that is a [valid integer](#). Positive numbers specify the relative position of the element's [focusable areas](#) in the [sequential focus navigation order](#), and negative numbers indicate that the control is to be unreachable by [sequential focus navigation](#).

Each element can have a **`tabindex focus flag`** set, as defined below. This flag is a factor that contributes towards determining whether an element is a [focusable area](#), as described in the previous section.

If the `tabindex` attribute is specified on an element, it must be parsed using the [rules for parsing integers](#). The attribute's values, or lack thereof, must be interpreted as follows:

If the attribute is omitted or parsing the value returns an error

The user agent should follow platform conventions to determine if the element's [tabindex focus flag](#) is set and, if so, whether the element and any [focusable areas](#) that have the element as their [DOM anchor](#) can be reached using [sequential focus navigation](#), and if so, what their relative position in the [sequential focus navigation order](#) is to be.

Modulo platform conventions, it is suggested that for the following elements, the [tabindex focus flag](#) be set:

- `<a>` elements that have an [href](#) attribute
- `<link>` elements that have an [href](#) attribute
- `<button>` elements
- `<input>` elements whose [type](#) attribute are not in the [Hidden](#) state
- `<select>` elements
- `<textarea>` elements
- `<menuitem>` elements
- Elements with a [draggable](#) attribute set, if that would enable the user agent to allow the user to begin a drag operations for those elements without the use of a pointing device
- [Editing hosts](#)
- [Browsing context containers](#)

NOTE:

One valid reason to ignore the platform conventions and always allow an element to be focused (by setting its [tabindex focus flag](#)) would be if the user's only mechanism for activating an element is through a keyboard action that triggers the focused element.

If the value is a negative integer

The user agent must set the element's [tabindex focus flag](#), but should omit the element from the [sequential focus navigation order](#).

NOTE:

One valid reason to ignore the requirement that sequential focus navigation not allow the author to lead to the element would be if the user's only mechanism for moving the focus is sequential focus navigation. For instance, a keyboard-only user would be unable to click on a text field with a negative `tabindex`, so that user's user agent would be well justified in allowing the user to tab to the control regardless.

If the value is a zero

The user agent must set the element's [tabindex focus flag](#), should allow the element and any [focusable areas](#) that have the element as their [DOM anchor](#) to be reached using [sequential focus navigation](#), following platform conventions to determine the element's relative position in the [sequential focus navigation order](#).

If the value is greater than zero

The user agent must set the element's [tabindex focus flag](#), should allow the element and any [focusable areas](#) that have the element as their [DOM anchor](#) to be reached using sequential focus navigation, and should place the element — referenced as *candidate* below — and the aforementioned [focusable areas](#) in the [sequential focus navigation order](#) so that, relative to other [focusable areas](#) in the [sequential focus navigation order](#), they are:

- before any [focusable area](#) whose [DOM anchor](#) is an element whose `tabindex` attribute has been omitted or whose value, when parsed, returns an error,
- before any [focusable area](#) whose [DOM anchor](#) is an element whose `tabindex` attribute has a value equal to or less than zero,
- after any [focusable area](#) whose [DOM anchor](#) is an element whose `tabindex` attribute has a value greater than zero but less than the value of the `tabindex` attribute on *candidate*,
- after any [focusable area](#) whose [DOM anchor](#) is an element whose `tabindex` attribute has a value equal to the value of the `tabindex` attribute on *candidate* but that is earlier in the document in [tree order](#) than *candidate*,
- before any [focusable area](#) whose [DOM anchor](#) is an element whose `tabindex` attribute has a value equal to the value of the `tabindex` attribute on *candidate* but that is later in the document in [tree order](#) than *candidate*, and
- before any [focusable area](#) whose [DOM anchor](#) is an element whose `tabindex` attribute has a value greater than the value of the `tabindex` attribute on *candidate*.

An element that has its [tabindex focus flag](#) set but does not otherwise have an [activation behavior](#) defined has an [activation behavior](#) that does nothing.

NOTE:

This means that an element that is only focusable because of its `tabindex` attribute will fire a `click` event in response to a non-mouse activation (e.g., hitting the "enter" key while the element is [focused](#)).

An element with the `tabindex` attribute specified is [interactive content](#).

The `tabIndex` IDL attribute must [reflect](#) the value of the `tabindex` content attribute. Its default value is 0 for elements that are focusable and -1 for elements that are not focusable.

⚠Warning! Most current browsers instead give the `tabIndex` IDL attribute a value of 0 for some list of elements that are by default a [focusable area](#), and -1 for other elements, if there is no [tabindex](#) content attribute set. This behaviour is not well-defined and will hopefully be improved in the future.

§ **5.4.4. Processing model**

The **focusing steps** for an object *new focus target* that is either a [focusable area](#), or an element that is not a [focusable area](#), or a [browsing context](#), are as follows. They can optionally be run with a *fallback target*.

1. If *new focus target* is not a [focusable area](#), then run the first matching set of steps from the following list:

↪ If *new focus target* is an `<area>` element with one or more shapes that are [focusable areas](#)

Let *new focus target* be the shape corresponding to the first `<img>` element in [tree order](#) that uses the image map to which the area element belongs.

↪ If *new focus target* is an element with one or more scrollable regions that are [focusable areas](#)

Let *new focus target* be the element's first scrollable region, according to a pre-order, depth-first traversal of the box tree. [\[CSS-2015\]](#)

↪ If *new focus target* is the [root element](#) of its Document

Let *new focus target* be the Document's [viewport](#).

↪ If *new focus target* is a [browsing context](#)

Let *new focus target* be the [browsing context](#)'s [active document](#).

↪ If *new focus target* is a [browsing context container](#)

Let *new focus target* be the [browsing context container](#)'s [nested browsing context](#)'s [active document](#).

↪ Otherwise

If no *fallback target* was specified, abort the [focusing steps](#).

Otherwise, let *new focus target* be the *fallback target*.

2. If *new focus target* is a [control group owner object](#) that is not a [focusable area](#), and its [control group](#) is not empty, then designate *new focus target* as the [focused area of the control group](#), and redo this step.

Otherwise, if *new focus target* is a [browsing context container](#), then let *new focus target* be the [nested browsing context](#)'s [active document](#), and redo this step.

3. If *new focus target* is a [focusable area](#) and its [DOM anchor](#) is [inert](#), then abort these steps.
4. If *new focus target* is the [currently focused area of a top-level browsing context](#), then abort these steps.
5. Let *old chain* be the [focus chain](#) of the [currently focused area of the top-level browsing context](#) in which *new focus target* finds itself.
6. Let *new chain* be the [focus chain](#) of *new focus target*.
7. Run the [focus update steps](#) with *old chain*, *new chain*, and *new focus target* respectively.

User agents must [immediately](#) run the [focusing steps](#) for a [focusable area](#), or [browsing context candidate](#) whenever the user attempts to move the focus to *candidate*.

The **unfocusing steps** for an object *old focus target* that is either a [focusable area](#) or an element that is not a [focusable area](#) are as follows:

1. If *old focus target* is [inert](#), then abort these steps.
 2. If *old focus target* is an `<area>` element and one of its shapes is the [currently focused area of a top-level browsing context](#), or, if *old focus target* is an element with one or more scrollable regions, and one of them is the [currently focused area of a top-level browsing context](#), then let *old focus target* be that [currently focused area of a top-level browsing context](#).
 3. Let *old chain* be the [focus chain](#) of the [currently focused area of a top-level browsing context](#).
 4. If *old focus target* is not one of the entries in *old chain*, then abort these steps.
 5. If *old focus target* is a [focusable area](#), then let *new focus target* be the first [focusable area](#) of its [control group](#) (if the [control group owner](#) is a [Document](#), this will always be a [viewport](#)).
- Otherwise, let *new focus target* be null.
6. If *new focus target* is not null, then run the [focusing steps](#) for *new focus target*.

When the [currently focused area of a top-level browsing context](#) is somehow unfocused without another element being explicitly focused in its stead, the user agent must [immediately](#) run the [unfocusing steps](#) for that object.

NOTE:

The [unfocusing steps](#) do not always result in the focus changing, even when applied to the [currently focused area of a top-level browsing context](#). For example, if the [currently focused area of a top-level browsing context](#) is a [viewport](#), then it will usually keep its focus regardless until another [focusable area](#) is explicitly focused with the [focusing steps](#).



When a [focusable area](#) is added to an empty [control group](#), it must be designated the [focused area of the control group](#).

Focus fixup rule one: When the designated [focused area of a control group](#) is removed from that [control group](#) in some way (e.g., it stops being a [focusable area](#), it is removed from the DOM, it becomes [expressly inert](#), etc), and the [control group](#) is still not empty: designate the first non-[inert](#) [focused area](#) in that [control group](#) to be the new [focused area of the control group](#), if any; if they are all [inert](#), then designate the first [focused area](#) in that [control group](#) to be the new [focused area of the control group](#) regardless of [inertness](#). If such a removal instead results in the [control group](#) being empty, then there is simply no longer a [focused area of the control group](#).

EXAMPLE 590

For example, this might happen because an element is removed from its [Document](#), or has a `hidden` attribute added. It might also happen to an `<input>` element when the element gets disabled.

When the [currently focused area of a top-level browsing context](#) was a [focusable area](#) but stops being a [focusable area](#), or when it starts being [inert](#), the user agent must run the following steps:

1. Let *old focus target* be whatever the [currently focused area of the top-level browsing context](#) was immediately before this algorithm became applicable (e.g., before the element was disabled, or the dialog was closed, or whatever caused this algorithm to run).
2. Let *old chain* be the [focus chain](#) of the [currently focused area of the top-level browsing context](#) at the same time.
3. Make sure that the changes implied by the focus fixup rules one, two, and three above are applied.
4. Let *new focus target* be the [currently focused area of a top-level browsing context](#).
5. If *old focus target* and *new focus target* are the same, abort these steps.
6. Let *new chain* be the [focus chain](#) of *new focus target*.
7. Run the [focus update steps](#) with *old chain*, *new chain*, and *new focus target* respectively.



The **focus update steps**, given an *old chain*, a *new chain*, and a *new focus target* respectively, are as follows:

1. Unset the [sequential focus navigation starting point](#).
2. If the last entry in *old chain* and the last entry in *new chain* are the same, pop the last entry from *old chain* and the last entry from *new chain* and redo this step.
3. For each entry *entry* in *old chain*, in order, run these substeps:
 1. If *entry* is an input element, and the change event [applies](#) to the element, and the element does not have a defined [activation behavior](#), and the user has changed the element's [value](#) or its list of [selected files](#) while the control was focused without committing that change, then [fire a simple event](#) that bubbles named `change` at the element.
 2. If *entry* is an element, let *blur event target* be *entry*.

If *entry* is a Document object, let *blur event target* be that Document object's Window object.

Otherwise, let *blur event target* be null.
 3. If *entry* is the last entry in *old chain*, and *entry* is an [Element](#), and the last entry in *new chain* is also an

[Element](#), then let *related blur target* be the last entry in *new chain*. Otherwise, let *related blur target* be null.

4. If *blur event target* is not null, [fire a focus event](#) named `blur` at *blur event target*, with *related blur target* as the related target.

NOTE:

In some cases, e.g., if *entry* is an area element's shape, a scrollable region, or a [viewport](#), no event is fired.

4. Apply any relevant platform-specific conventions for focusing *new focus target*. (For example, some platforms select the contents of a text field when that field is focused.)

5. For each entry *entry* in *new chain*, in reverse order, run these substeps:

1. If *entry* is a [focusable area](#): Designate *entry* as the [focused area of the control group](#).
2. If *entry* is an element, let *focus event target* be *entry*.

If *entry* is a Document object, let *focus event target* be that Document object's Window object.

Otherwise, let *focus event target* be null.

3. If *entry* is the last entry in *new chain*, and *entry* is an [Element](#), and the last entry in *old chain* is also an [Element](#), then let *related focus target* be the last entry in *old chain*. Otherwise, let *related focus target* be null.
4. If *focus event target* is not null, [fire a focus event](#) named `focus` at *focus event target*, with *related focus target* as the related target.

NOTE:

In some cases, e.g., if *entry* is an area element's shape, a scrollable region, or a [viewport](#), no event is fired.

When a user agent is required to **fire a focus event** named *e* at an element *t* and with a given related target *r*, the user agent must create a [trusted](#) FocusEvent object, initialize it to have the given name *e*, to not bubble, to not be cancelable, and to have the `relatedTarget` attribute initialized to *r*, the `view` attribute initialized to the Window object of the Document object of *t*, and the `detail` attribute initialized to 0, and must then [dispatch](#) the newly created FocusEvent object at the specified target element *t*.



When a key event is to be routed in a [top-level browsing context](#), the user agent must run the following steps:

1. Let *target area* be the [currently focused area of the top-level browsing context](#).
2. If *target area* is a [focusable area](#), let *target node* be *target area*'s [DOM anchor](#).
3. If *target node* is a Document that has a `<body>` element, then let *target node* be the [body element](#) of that [Document](#).
Otherwise, if *target node* is a Document that has a [root element](#), then let *target node* be the [root element](#) of that [Document](#).
4. If *target node* is not [inert](#), fire the event at *target node*.

NOTE:

It is possible for the [currently focused area of a top-level browsing context](#) to be [inert](#). It is likely to be the result of a logic error in the application, though.

5. If the event was not canceled, then let *target area* handle the key event. This might include [running synthetic click activation steps](#) for *target node*.



The **has focus steps**, given a Document object *target*, are as follows:

1. Let *candidate* be the Document of the [top-level browsing context](#).
2. If *candidate* is *target*, return true and abort these steps.
3. If *candidate* has a non-empty [control group](#), and the designated [focused area of the control group](#) is a [browsing context container](#), and the [active document](#) of that [browsing context container](#)'s [nested browsing context](#) is *target*, then return true and abort these steps.

Otherwise, if *candidate* has a non-empty [control group](#), and the designated [focused area of the control group](#) is a [browsing context container](#), then let *candidate* be the [active document](#) of that [browsing context container](#)'s [nested browsing context](#), and redo this step.

Otherwise, return false and abort these steps.

§ 5.4.5. Sequential focus navigation

Each [control group](#) has a **sequential focus navigation order**, which orders some or all of the [focusable areas](#) in the [control group](#) relative to each other. The order in the [sequential focus navigation order](#) does not have to be related to the order in the [control group](#) itself. If a [focusable area](#) is omitted from the [sequential focus navigation order](#) of its [control group](#), then it is unreachable via [sequential focus navigation](#).

There can also be a **sequential focus navigation starting point**. It is initially unset. The user agent may set it when the user indicates that it should be moved.

EXAMPLE 591

For example, the user agent could set it to the position of the user's click if the user clicks on the document contents.

When the user requests that focus move from the [currently focused area of a top-level browsing context](#) to the next or previous [focusable area](#) (e.g., as the default action of pressing the `tab` key), or when the user requests that focus sequentially move to a [top-level browsing context](#) in the first place (e.g., from the browser's location bar), the user agent must use the following algorithm:

1. Let *starting point* be the [currently focused area of a top-level browsing context](#), if the user requested to move focus sequentially from there, or else the [top-level browsing context](#) itself, if the user instead requested to move focus from outside the [top-level browsing context](#).
2. If there is a [sequential focus navigation starting point](#) defined and it is inside *starting point*, then let *starting point* be the [sequential focus navigation starting point](#) instead.
3. Let *direction* be *forward* if the user requested the *next* control, and *backward* if the user requested the previous control.

NOTE:

Typically, pressing `tab` requests the next control, and pressing `shift+tab` requests the previous control.

4. *Loop*: Let *selection mechanism* be *sequential* if the *starting point* is a [browsing context](#) or if *starting point* is in its [control group](#)'s [sequential focus navigation order](#).

Otherwise, *starting point* is not in its [control group](#)'s [sequential focus navigation order](#); let *selection mechanism* be *DOM*.

5. Let *candidate* be the result of running the [sequential navigation search algorithm](#) with *starting point*, *direction*, and *selection mechanism* as the arguments.
6. If *candidate* is not null, then run the [focusing steps](#) for *candidate* and abort these steps.
7. Otherwise, unset the [sequential focus navigation starting point](#).
8. If *starting point* is the [top-level browsing context](#), or a [focusable area](#) in the [top-level browsing context](#), the user agent should transfer focus to its own controls appropriately (if any), honouring *direction*, and then abort these steps.

EXAMPLE 592

For example, if *direction* is *backward*, then the last focusable control before the browser's rendering area would be the control to focus.

If the user agent has no focusable controls — a kiosk-mode browser, for instance — then the user agent may instead restart these steps with the *starting point* being the [top-level browsing context](#) itself.

- Otherwise, *starting point* is a [focusable area](#) in a [nested browsing context](#). Let *starting point* be that [nested browsing context](#)'s [browsing context container](#), and return to the step labeled *loop*.

The **sequential navigation search algorithm** consists of the following steps. This algorithm takes three arguments: *starting point*, *direction*, and *selection mechanism*.

- Pick the appropriate cell from the following table, and follow the instructions in that cell.

The appropriate cell is the one that is from the column whose header describes *direction* and from the first row whose header describes *starting point* and *selection mechanism*.

	<i>direction</i> is <i>forward</i>	<i>direction</i> is <i>backward</i>
<i>starting point</i> is a browsing context	Let <i>candidate</i> be the first suitable sequentially focusable area in <i>starting point</i> 's active document 's primary control group , if any; or else null	Let <i>candidate</i> be the last suitable sequentially focusable area in <i>starting point</i> 's active document 's primary control group , if any; or else null
<i>selection mechanism</i> is <i>DOM</i>	Let <i>candidate</i> be the first suitable sequentially focusable area in the home control group following <i>starting point</i> , if any; or else null	Let <i>candidate</i> be the last suitable sequentially focusable area in the home control group preceding <i>starting point</i> , if any; or else null
<i>selection mechanism</i> is <i>sequential</i>	Let <i>candidate</i> be the first suitable sequentially focusable area in the home sequential focus navigation order following <i>starting point</i> , if any; or else null	Let <i>candidate</i> be the last suitable sequentially focusable area in the home sequential focus navigation order preceding <i>starting point</i> , if any; or else null

A **suitable sequentially focusable area** is a [focusable area](#) whose [DOM anchor](#) is not [inert](#) and that is in its [control group](#)'s [sequential focus navigation order](#).

The **primary control group** of a [control group owner object](#) *X* is the [control group](#) of *X*.

The **home control group** is the [control group](#) to which *starting point* belongs.

The **home sequential focus navigation order** is the [sequential focus navigation order](#) to which *starting point* belongs.

NOTE:

The [home sequential focus navigation order](#) is the [home control group](#)'s [sequential focus navigation order](#), but is only used when the *starting point* is in that [sequential focus navigation order](#) (when it's not, *selection mechanism* will be *DOM*).

- If *candidate* is a [browsing context container](#), then let *new candidate* be the result of running the [sequential navigation search algorithm](#) with *candidate*'s [nested browsing context](#) as the first argument, *direction* as the second, and *sequential* as the third.

If *new candidate* is null, then let *starting point* be *candidate*, and return to the top of this algorithm. Otherwise, let *candidate* be *new candidate*.

- Return *candidate*.

§ 5.4.6. Focus management APIs

This definition is non-normative. Implementation requirements are given below this definition.

document*.*activeElement

Returns the deepest element in the document through which or to which key events are being routed. This is,

roughly speaking, the focused element in the document.

For the purposes of this API, when a [child browsing context](#) is focused, its [browsing context container](#) is [focused](#) in the [parent browsing context](#). For example, if the user moves the focus to a text field in an `<iframe>`, the `iframe` is the element returned by the `activeElement` API in the `iframe`'s [node document](#).

`document . hasFocus()`

Returns true if key events are being routed through or to the document; otherwise, returns false. Roughly speaking, this corresponds to the document, or a document nested inside this one, being focused.

`window . focus()`

Moves the focus to the window's [browsing context](#), if any.

`element . focus()`

Moves the focus to the element.

If the element is a [browsing context container](#), moves the focus to the [nested browsing context](#) instead.

`element . blur()`

Moves the focus to the [viewport](#). Use of this method is discouraged; if you want to focus the [viewport](#), call the `focus()` method on the Document's root element.

Do not use this method to hide the focus ring if you find the focus ring unsightly. Instead, use a CSS rule to override the `'outline'` property, and provide a different way to show what element is focused. Be aware that if an alternative focusing style isn't made available, the page will be significantly less usable for people who primarily navigate pages using a keyboard, or those with reduced vision who use focus outlines to help them navigate the page.

EXAMPLE 593

For example, to hide the outline from links and instead use a yellow background to indicate focus, you could use:

```
:link:focus, :visited:focus { outline: none; background: yellow; color: black; }
```

Do not use this method to hide the focus ring. Do not use any other method that hides the focus ring from keyboard users, in particular do not use a CSS rule to override the `'outline'` property. Removal of the focus ring leads to serious accessibility issues for users who navigate and interact with interactive content using the keyboard.

The **`activeElement`** attribute on Document objects must return the value returned by the following steps:

1. Let *candidate* be the Document on which the method was invoked.
2. If *candidate* has a non-empty [control group](#), let *candidate* be the designated [focused area of the control group](#).
3. If *candidate* is a [focusable area](#), let *candidate* be *candidate*'s [DOM anchor](#).
4. If *candidate* is a Document that has a `<body>` element, then let *candidate* be the [body element](#) of that [Document](#).

Otherwise, if *candidate* is a Document that has a [root element](#), then let *candidate* be the [root element](#) of that [Document](#).

Otherwise, if *candidate* is a [Document](#), then let *candidate* be null.

5. Return *candidate*.

The **`hasFocus()`** method on the Document object, when invoked, must return the result of running the [has focus steps](#) with the Document object as the argument.

The **`focus()`** method on the Window object, when invoked, must run the [focusing steps](#) with the Window object's [browsing context](#). Additionally, if this [browsing context](#) is a [top-level browsing context](#), user agents are encouraged to trigger some sort of notification to indicate to the user that the page is attempting to gain focus.

The **`blur()`** method on the Window object, when invoked, provides a hint to the user agent that the script believes the user probably is not currently interested in the contents of the [browsing context](#) of the Window object on which the method was

invoked, but that the contents might become interesting again in the future.

User agents are encouraged to ignore calls to this `blur()` method entirely.

NOTE:

Historically, the `focus()` and `blur()` methods actually affected the system-level focus of the system widget (e.g., tab or window) that contained the [browsing context](#), but hostile sites widely abuse this behavior to the user's detriment.

The `focus()` method on elements, when invoked, must run the following algorithm:

1. If the element is marked as *locked for focus*, then abort these steps.
2. Mark the element as **locked for focus**.
3. Run the [focusing steps](#) for the element.
4. Unmark the element as *locked for focus*.

The `blur()` method, when invoked, should run the [unfocusing steps](#) for the element on which the method was called. User agents may selectively or uniformly ignore calls to this method for usability reasons.

EXAMPLE 594

For example, if the `blur()` method is unwisely being used to remove the focus ring for aesthetics reasons, the page would become unusable by keyboard users. Ignoring calls to this method would thus allow keyboard users to interact with the page.

§ 5.5. Assigning keyboard shortcuts

§ 5.5.1. Introduction

This section is non-normative.

Each element that can be activated or focused can be assigned a shortcut key combination to activate it, using the [accesskey](#) attribute.

The exact shortcut is determined by the user agent, potentially using information about the user's preferences, what keyboard shortcuts already exist on the platform, and what other shortcuts have been specified on the page, as well as the value of the [accesskey](#) attribute.

A valid value for [accesskey](#) consists of a single character, such as a letter or digit.

User agents can provide users with a list of the keyboard shortcuts, but authors are encouraged to do so also.

EXAMPLE 595

In this example, an author has provided a button that can be invoked using a shortcut key, and suggested "C" as a memorable and useful shortcut.

```
<input type=button value=Collect onclick="collect()" accesskey="C" id=c>
```

§ 5.5.2. The [accesskey](#) attribute

All [html elements](#) may have the **accesskey** content attribute set. The [accesskey](#) attribute's value is used by the user agent as a guide for creating a keyboard shortcut that activates or focuses the element.

If specified, the value must be a single printable character: a string exactly one Unicode code point in length.

Authors should not use " ", nor characters that normally require a modifier key to generate, as a value of [accesskey](#).

EXAMPLE 596

In the following example, a variety of links are given with access keys so that keyboard users familiar with the site can more quickly navigate to the relevant pages:

```
<nav>
  <p>
    <a title="Consortium Activities" accesskey="A" href="/Consortium
/activities">Activities</a> |
    <a title="Technical Reports and Recommendations" accesskey="T" href="/TR/">Technical
Reports</a> |
    <a title="Alphabetical Site Index" accesskey="S" href="/Consortium/siteindex">Site
Index</a> |
    <a title="About This Site" accesskey="B" href="/Consortium/">About Consortium</a> |
    <a title="Contact Consortium" accesskey="C" href="/Consortium/contact">Contact</a>
  </p>
</nav>
```

§ 5.5.3. Processing model

An element's **assigned access key** is a key combination derived from the element's [accesskey](#) content attribute, or assigned by the user agent, optionally based on a user preference. Initially, an element must not have an [assigned access key](#).

Whenever an element's [accesskey](#) attribute is set, changed, or removed, the user agent must update the element's [assigned access key](#) by running the following steps:

1. If the element has no [accesskey](#) attribute, then skip to the *fallback* step below.
2. The user agent may assign a key combination based on stored user preferences as the element's [assigned access key](#) and then abort these steps.
3. Let *value* be the value of the [accesskey](#) attribute.
4. The user agent may strip content from *value* to reduce the length of *value* to a single unicode code point.
5. If *value* is not a string exactly one Unicode code point in length, then abort these steps.
6. The user agent may assign a combination of a mix of zero or more modifier keys and *value* as the element's [assigned access key](#) and abort these steps.
7. *Fallback*: Optionally, the user agent may assign a key combination of its choosing as the element's [assigned access key](#) and then abort these steps.
8. If this step is reached, the element has no [assigned access key](#).

Once a user agent has selected and assigned an access key for an element, the user agent should not change the element's [assigned access key](#) unless the [accesskey](#) content attribute is changed or the element is moved to another [Document](#).

When the user presses the key combination corresponding to the [assigned access key](#) for an element, if the element defines a command, the command's [Hidden State](#) facet is false (visible), the command's [Disabled State](#) facet is also false (enabled), the element is [in a Document](#) that has an associated [browsing context](#), and neither the element nor any of its ancestors has a hidden attribute specified, then the user agent must trigger the [Action](#) of the command.

NOTE:

User agents might expose elements that have an [accesskey](#) attribute in other ways as well, e.g., in a menu displayed in response to a specific key combination.



The `accessKey` IDL attribute must [reflect](#) the [accesskey](#) content attribute.

§ 5.6. Editing

§ 5.6.1. Making document regions editable: The `contentEditable` content attribute

```
[NoInterfaceObject]
interface ElementContentEditable {
  attribute DOMString contentEditable;
  readonly attribute boolean isContentEditable;
};
```

The **contentEditable** content attribute is an [enumerated attribute](#) whose keywords are the empty string, `true`, and `false`. The empty string and the `true` keyword map to the *true* state. The `false` keyword maps to the *false* state. In addition, there is a third state, the *inherit* state, which is the *missing value default* (and the *invalid value default*).

The *true* state indicates that the element is editable. The *inherit* state indicates that the element is editable if its parent is. The *false* state indicates that the element is not editable.

This definition is non-normative. Implementation requirements are given below this definition.

`element . contentEditable` [= *value*]

Returns "true", "false", or "inherit", based on the state of the [contentEditable](#) attribute.

Can be set, to change that state.

Throws a "[SyntaxError](#)" [DOMException](#) if the new value isn't one of those strings.

`element . isContentEditable`

Returns true if the element is editable; otherwise, returns false.

The **contentEditable** IDL attribute, on getting, must return the string "true" if the content attribute is set to the true state, "false" if the content attribute is set to the false state, and "inherit" otherwise. On setting, if the new value is an [ASCII case-insensitive](#) match for the string "inherit" then the content attribute must be removed, if the new value is an [ASCII case-insensitive](#) match for the string "true" then the content attribute must be set to the string "true", if the new value is an [ASCII case-insensitive](#) match for the string "false" then the content attribute must be set to the string "false", and otherwise the attribute setter must throw a "[SyntaxError](#)" [DOMException](#).

The **isContentEditable** IDL attribute, on getting, must return true if the element is either an [editing host](#) or [editable](#), and false otherwise.

§ 5.6.2. Making entire documents editable: The `designMode` IDL attribute

Documents have a **designMode**, which can be either enabled or disabled.

This definition is non-normative. Implementation requirements are given below this definition.

`document . designMode` [= *value*]

Returns "on" if the document is editable, and "off" if it isn't.

Can be set, to change the document's current state. This focuses the document and resets the selection in that document.

The **designMode** IDL attribute on the Document object takes two values, "on" and "off". On setting, the new value must be compared in an [ASCII case-insensitive](#) manner to these two values; if it matches the "on" value, then **designMode** must be enabled, and if it matches the "off" value, then **designMode** must be disabled. Other values must be ignored.

On getting, if **designMode** is enabled, the IDL attribute must return the value "on"; otherwise it is disabled, and the attribute must return the value "off".

The last state set must persist until the document is destroyed or the state is changed. Initially, documents must have their **designMode** disabled.

When the **designMode** changes from being disabled to being enabled, the user agent must [immediately](#) reset the document's

[active range](#)'s start and end boundary points to be at the start of the Document and then run the [focusing steps](#) for the root element of the [Document](#), if any.

§ 5.6.3. Best practices for in-page editors

Authors are encouraged to set the `'white-space'` property on [editing hosts](#) and on markup that was originally created through these editing mechanisms to the value `'pre-wrap'`. Default HTML whitespace handling is not well suited to WYSIWYG editing, and line wrapping will not work correctly in some corner cases if `'white-space'` is left at its default value.

EXAMPLE 597

As an example of problems that occur if the default `'normal'` value is used instead, consider the case of the user typing "yellow_ball", with two spaces (here represented by "␣") between the words. With the editing rules in place for the default value of `'white-space'` (`'normal'`), the resulting markup will either consist of "yellow ball" or "yellow ball"; i.e., there will be a non-breaking space between the two words in addition to the regular space. This is necessary because the `'normal'` value for `'white-space'` requires adjacent regular spaces to be collapsed together.

In the former case, "yellow␣" might wrap to the next line ("␣" being used here to represent a non-breaking space) even though "yellow" alone might fit at the end of the line; in the latter case, "␣ball", if wrapped to the start of the line, would have visible indentation from the non-breaking space.

When `'white-space'` is set to `'pre-wrap'`, however, the editing rules will instead simply put two regular spaces between the words, and should the two words be split at the end of a line, the spaces would be neatly removed from the rendering.

§ 5.6.4. Editing APIs

The definition of the terms **active range**, **editing host**, and **editable**, the user interface requirements of elements that are [editing hosts](#) or [editable](#), the `'execCommand()'`, `'queryCommandEnabled()'`, `'queryCommandIndeterm()'`, `'queryCommandState()'`, `'queryCommandSupported()'`, and `'queryCommandValue()'` methods, text selections, and the **delete the selection** algorithm are being specified in the various developing HTML Editing specification drafts [\[EDITING\]](#). The interaction of editing and undo/redo features are being specified in the UndoManager and DOM Transaction specification. [\[UNDO\]](#)

§ 5.6.5. Spelling and grammar checking

User agents can support the checking of spelling and grammar of editable text, either in form controls (such as the value of `<textarea>` elements), or in elements in an [editing host](#) (e.g., using [contenteditable](#)).

For each element, user agents must establish a **default behavior**, either through defaults or through preferences expressed by the user. There are three possible default behaviors for each element:

true-by-default

The element will be checked for spelling and grammar if its contents are editable and spellchecking is not explicitly disabled through the [spellcheck](#) attribute.

false-by-default

The element will never be checked for spelling and grammar unless spellchecking is explicitly enabled through the [spellcheck](#) attribute.

inherit-by-default

The element's default behavior is the same as its parent element's. Elements that have no parent element cannot have this as their default behavior.



The **spellcheck** attribute is an [enumerated attribute](#) whose keywords are the empty string, `true` and `false`. The empty string and the `true` keyword map to the *true* state. The `false` keyword maps to the *false* state. In addition, there is a third state, the *default* state, which is the *missing value default* (and the *invalid value default*).

NOTE:

The *true* state indicates that the element is to have its spelling and grammar checked. The default state indicates that the element is to act according to a default behavior, possibly based on the parent element's own `spellcheck` state, as defined below. The *false* state indicates that the element is not to be checked.



This definition is non-normative. Implementation requirements are given below this definition.

***element* . `spellcheck` [= *value*]**

Returns true if the element is to have its spelling and grammar checked; otherwise, returns false.

Can be set, to override the default and set the `spellcheck` content attribute.

***element* . `forceSpellCheck()`**

Forces the user agent to report spelling and grammar errors on the element (if checking is enabled), even if the user has never focused the element. (If the method is not invoked, user agents can hide errors in text that wasn't just entered by the user.)

The `spellcheck` IDL attribute, on getting, must return true if the element's `spellcheck` content attribute is in the *true* state, or if the element's `spellcheck` content attribute is in the *default* state and the element's `default behavior` is *true-by-default*, or if the element's `spellcheck` content attribute is in the *default* state and the element's `default behavior` is *inherit-by-default* and the element's parent element's `spellcheck` IDL attribute would return true; otherwise, if none of those conditions applies, then the attribute must instead return false.

NOTE:

The `spellcheck` IDL attribute is not affected by user preferences that override the `spellcheck` content attribute, and therefore might not reflect the actual spellchecking state.

On setting, if the new value is true, then the element's `spellcheck` content attribute must be set to the literal string "true", otherwise it must be set to the literal string "false".



User agents must only consider the following pieces of text as checkable for the purposes of this feature:

- The *value* of `<input>` elements whose *type* attributes are in the *Text*, *Search*, *URL*, or *E-mail* states and that are *mutable* (i.e., that do not have the `readonly` attribute specified and that are not disabled).
- The *value* of `<textarea>` elements that do not have a `readonly` attribute and that are not disabled.
- Text in Text nodes that are children of *editing hosts* or *editable* elements.
- Text in attributes of *editable* elements.

For text that is part of a Text node, the element with which the text is associated is the element that is the immediate parent of the first character of the word, sentence, or other piece of text. For text in attributes, it is the attribute's element. For the values of input and `<textarea>` elements, it is the element itself.

To determine if a word, sentence, or other piece of text in an applicable element (as defined above) is to have spelling- and grammar-checking enabled, the user agent must use the following algorithm:

1. If the user has disabled the checking for this text, then the checking is disabled.
2. Otherwise, if the user has forced the checking for this text to always be enabled, then the checking is enabled.
3. Otherwise, if the element with which the text is associated has a `spellcheck` content attribute, then: if that attribute is in the *true* state, then checking is enabled; otherwise, if that attribute is in the *false* state, then checking is disabled.
4. Otherwise, if there is an ancestor element with a `spellcheck` content attribute that is not in the *default* state, then: if the

nearest such ancestor's `spellcheck` content attribute is in the *true* state, then checking is enabled; otherwise, checking is disabled.

5. Otherwise, if the element's `default behavior` is *true-by-default*, then checking is enabled.
6. Otherwise, if the element's `default behavior` is *false-by-default*, then checking is disabled.
7. Otherwise, if the element's parent element has *its* checking enabled, then checking is enabled.
8. Otherwise, checking is disabled.

If the checking is enabled for a word/sentence/text, the user agent should indicate spelling and grammar errors in that text. User agents should take into account the other semantics given in the document when suggesting spelling and grammar corrections. User agents may use the language of the element to determine what spelling and grammar rules to use, or may use the user's preferred language settings. user agents should use `<input>` element attributes such as `pattern` to ensure that the resulting value is valid, where possible.

If checking is disabled, the user agent should not indicate spelling or grammar errors for that text.

Even when checking is enabled, user agents may opt to not report spelling or grammar errors in text that the user agent deems the user has no interest in having checked (e.g., text that was already present when the page was loaded, or that the user did not type, or text in controls that the user has not focused, or in parts of e-mail addresses that the user agent is not confident were misspelt). The `'forceSpellcheck()'` method, when invoked on an element, must override this behavior, forcing the user agent to consider all spelling and grammar errors in text in that element for which checking is enabled to be of interest to the user.

EXAMPLE 598

The element with ID "a" in the following example would be the one used to determine if the word "Hello" is checked for spelling errors. In this example, it would not be.

```
<div contenteditable="true">
  <span spellcheck="false">Hell</span><em>o!</em>
</div>
```

The element with ID "b" in the following example would have checking enabled (the leading space character in the attribute's value on the `<input>` element causes the attribute to be ignored, so the ancestor's value is used instead, regardless of the default).

```
<p spellcheck="true"><label>Name: <input spellcheck=" false"></label>
</p>
```

NOTE:

This specification does not define the user interface for spelling and grammar checkers. A user agent could offer on-demand checking, could perform continuous checking while the checking is enabled, or could use other interfaces.

§ 5.7. Drag and drop

This section defines an event-based drag-and-drop mechanism.

This specification does not define exactly what a *drag-and-drop operation* actually is.

On a visual medium with a pointing device, a drag operation could be the default action of a `mousedown` event that is followed by a series of `mousemove` events, and the drop could be triggered by the mouse being released.

When using an input modality other than a pointing device, users would probably have to explicitly indicate their intention to perform a drag-and-drop operation, stating what they wish to drag and where they wish to drop it, respectively.

However it is implemented, drag-and-drop operations must have a starting point (e.g., where the mouse was clicked, or the start of the selection or element that was selected for the drag), may have any number of intermediate steps (elements that the mouse moves over during a drag, or elements that the user picks as possible drop points as he cycles through possibili-

ties), and must either have an end point (the element above which the mouse button was released, or the element that was finally selected), or be canceled. The end point must be the last element selected as a possible drop point before the drop occurs (so if the operation is not canceled, there must be at least one element in the middle step).

§ 5.7.1. Introduction

This section is non-normative.

To make an element draggable is simple: give the element a [draggable](#) attribute, and set an event listener for `dragstart` that stores the data being dragged.

The event handler typically needs to check that it's not a text selection that is being dragged, and then needs to store data into the [DataTransfer](#) object and set the allowed effects (copy, move, link, or some combination).

For example:

```
<p>What fruits do you like?</p>
<ol ondragstart="dragStartHandler(event)">
  <li draggable="true">Apples</li>
  <li draggable="true">Oranges</li>
  <li draggable="true">Pears</li>
</ol>
<script>
var internalDNDType = 'text/x-example'; // set this to something specific to your site
function dragStartHandler(event) {
  if (event.target instanceof HTMLLIElement) {
    // use the element's
```



To accept a drop, the drop target has to have a [dropzone](#) attribute and listen to the drop event.

The value of the [dropzone](#) attribute specifies what kind of data to accept (e.g., `"string:text/plain"` to accept any text strings, or `"file:image/png"` to accept a PNG image file) and what kind of feedback to give (e.g., `"move"` to indicate that the data will be moved).

NOTE:

Instead of using the [dropzone](#) attribute, a drop target can handle the `dragenter` event (to report whether or not the drop target is to accept the drop) and the `dragover` event (to specify what feedback is to be shown to the user).

The drop event allows the actual drop to be performed. This event needs to be canceled, so that the [dropEffect](#) attribute's value can be used by the source (otherwise it's reset).

For example:

```

<p>Drop your favorite fruits below:</p>
<ol dropzone="move string:text/x-example" ondrop="dropHandler(event)">
  <!-- don't forget to change the "text/x-example" type to something
specific to your site -->
</ol>
<script>
var internalDNDType = 'text/x-example'; // set this to something specific to your site
function dropHandler(event) {
  var li = document.createElement('li');
  var data = event.dataTransfer.getData(internalDNDType);
  if (data == 'fruit-apple') {
    li.textContent = 'Apples';
  } else if (data == 'fruit-orange') {
    li.textContent = 'Oranges';
  } else if (data == 'fruit-pear') {
    li.textContent = 'Pears';
  } else {
    li.textContent = 'Unknown Fruit';
  }
  event.target.appendChild(li);
}
</script>

```



To remove the original element (the one that was dragged) from the display, the `dragend` event can be used.

For our example here, that means updating the original markup to handle that event:

```

<p>What fruits do you like?</p>
<ol ondragstart="dragStartHandler(event)" ondragend="dragEndHandler(event)">
  <!-- ...as before... -->
</ol>
<script>
function dragStartHandler(event) {
  // ...as before...
}
function dragEndHandler(event) {
  if (event.dataTransfer.dropEffect == 'move') {
    // remove the dragged element
    event.target.parentNode.removeChild(event.target);
  }
}
</script>

```

§ 5.7.2. The drag data store

The data that underlies a drag-and-drop operation, known as the **drag data store**, consists of the following information:

- A **drag data store item list**, which is a list of items representing the dragged data, each consisting of the following information:

The drag data item kind

The kind of data:

Plain Unicode string

Text.

File

Binary data with a file name.

The drag data item type string

A Unicode string giving the type or format of the data, generally given by a [MIME type](#). Some values that are not [MIME types](#) are special-cased for legacy reasons. The API does not enforce the use of [MIME types](#); other values can be used as well. In all cases, however, the values are all [converted to ASCII lowercase](#) by the API.

NOTE:

Strings that contain [space characters](#) cannot be used with the [dropzone](#) attribute, so authors are encouraged to use only [MIME types](#) or custom strings (without spaces).

There is a limit of one *Plain Unicode string* item per [item type string](#).

The actual data

A Unicode or binary string, in some cases with a file name (itself a Unicode string), as per [the drag data item kind](#).

The [drag data store item list](#) is ordered in the order that the items were added to the list; most recently added last.

- The following information, used to generate the UI feedback during the drag:
 - User-agent-defined default feedback information, known as the **drag data store default feedback**.
 - Optionally, a bitmap image and the coordinate of a point within that image, known as the **drag data store bitmap** and **drag data store hot spot coordinate**.
- A **drag data store mode**, which is one of the following:

Read/write mode

For the dragstart event. New data can be added to the [drag data store](#).

Read-only mode

For the drop event. The list of items representing dragged data can be read, including the data. No new data can be added.

Protected mode

For all other events. The formats and kinds in the [drag data store](#) list of items representing dragged data can be enumerated, but the data itself is unavailable and no new data can be added.

- A **drag data store allowed effects state**, which is a string.

When a [drag data store](#) is **created**, it must be initialized such that its [drag data store item list](#) is empty, it has no [drag data store default feedback](#), it has no [drag data store bitmap](#) and [drag data store hot spot coordinate](#), its [drag data store mode](#) is [protected mode](#), and its [drag data store allowed effects state](#) is the string "uninitialized".

§ 5.7.3. The `DataTransfer` interface

[DataTransfer](#) objects are used to expose the [drag data store](#) that underlies a drag-and-drop operation.

```
interface DataTransfer {
  attribute DOMString dropEffect;
  attribute DOMString effectAllowed;

  [SameObject] readonly attribute DataTransferItemList items;

  void setDragImage(Element image, long x, long y);

  /* old interface */
  [SameObject] readonly attribute DOMString[] types;
  DOMString getData(DOMString format);
  void setData(DOMString format, DOMString data);
  void clearData(optional DOMString format);
  [SameObject] readonly attribute FileList files;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

`dataTransfer . dropEffect [ = value ]`

Returns the kind of operation that is currently selected. If the kind of operation isn't one of those that is allowed by the [effectAllowed](#) attribute, then the operation will fail.

Can be set, to change the selected operation.

The possible values are "none", "copy", "link", and "move".

`dataTransfer . effectAllowed [ = value ]`

Returns the kinds of operations that are to be allowed.

Can be set (during the dragstart event), to change the allowed operations.

The possible values are "none", "copy", "copyLink", "copyMove", "link", "linkMove", "move", "all", and "uninitialized",

`dataTransfer . items`

Returns a [DataTransferItemList](#) object, with the drag data.

`dataTransfer . setDragImage( element , x , y )`

Uses the given element to update the drag feedback, replacing any previously specified feedback.

`dataTransfer . types`

Returns an array listing the formats that were set in the dragstart event. In addition, if any files are being dragged, then one of the types will be the string "Files".

`data = dataTransfer . getData( format )`

Returns the specified data. If there is no such data, returns the empty string.

`dataTransfer . setData( format , data )`

Adds the specified data.

`dataTransfer . clearData( [ format ] )`

Removes the data of the specified formats. Removes all data if the argument is omitted.

`dataTransfer . files`

Returns a [FileList](#) of the files being dragged, if any.

[DataTransfer](#) objects are used during the [drag-and-drop events](#), and are only valid while those events are being fired.

A [DataTransfer](#) object is associated with a [drag data store](#) while it is valid.

The **dropEffect** attribute controls the drag-and-drop feedback that the user is given during a drag-and-drop operation. When the [DataTransfer](#) object is created, the [dropEffect](#) attribute is set to a string value. On getting, it must return its current value. On setting, if the new value is one of "none", "copy", "link", or "move", then the attribute's current value must be set to the new value. Other values must be ignored.

The **effectAllowed** attribute is used in the drag-and-drop processing model to initialize the [dropEffect](#) attribute during the dragenter and dragover events. When the [DataTransfer](#) object is created, the [effectAllowed](#) attribute is set to a string value. On getting, it must return its current value. On setting, if [drag data store](#)'s mode is the [read/write mode](#) and the new value is one of "none", "copy", "copyLink", "copyMove", "link", "linkMove", "move", "all", or "uninitialized", then the attribute's current value must be set to the new value. Otherwise it must be left unchanged.

The **items** attribute must return a [DataTransferItemList](#) object associated with the [DataTransfer](#) object.

The **setDragImage(element , x , y)** method must run the following steps:

1. If the [DataTransfer](#) object is no longer associated with a [drag data store](#), abort these steps. Nothing happens.
2. If the [drag data store](#)'s mode is not the [read/write mode](#), abort these steps. Nothing happens.
3. If the *element* argument is an `<img>` element, then set the [drag data store bitmap](#) to the element's image (at its [intrinsic size](#)); otherwise, set the [drag data store bitmap](#) to an image generated from the given element (the exact mechanism for doing so is not currently specified).

4. Set the [drag data store hot spot coordinate](#) to the given x , y coordinate.

The **types** attribute must return a [live read only](#) array giving the strings that the following steps would produce.

1. Start with an empty list L .
2. If the [DataTransfer](#) object is no longer associated with a [drag data store](#), the array is empty. Abort these steps; return the empty list L .
3. For each item in the [drag data store item list](#) whose **kind** is *Plain Unicode string*, add an entry to the list L consisting of the item's [type string](#).
4. If there are any items in the [drag data store item list](#) whose **kind** is *File*, then add an entry to the list L consisting of the string "Files". (This value can be distinguished from the other values because it is not lowercase.)
5. The strings produced by these steps are those in the list L .

The **getData(format)** method must run the following steps:

1. If the [DataTransfer](#) object is no longer associated with a [drag data store](#), return the empty string and abort these steps.
2. If the [drag data store](#)'s mode is the [protected mode](#), return the empty string and abort these steps.
3. Let *format* be the first argument, [converted to ASCII lowercase](#).
4. Let *convert-to-URL* be false.
5. If *format* equals "text", change it to "text/plain".
6. If *format* equals "url", change it to "text/uri-list" and set *convert-to-URL* to true.
7. If there is no item in the [drag data store item list](#) whose **kind** is *Plain Unicode string* and whose [type string](#) is equal to *format*, return the empty string and abort these steps.
8. Let *result* be the data of the item in the [drag data store item list](#) whose **kind** is *Plain Unicode string* and whose [type string](#) is equal to *format*.
9. If *convert-to-URL* is true, then parse *result* as appropriate for text/uri-list data, and then set *result* to the first URL from the list, if any, or the empty string otherwise. [\[RFC2483\]](#)
10. Return *result*.

The **setData(format , data)** method must run the following steps:

1. If the [DataTransfer](#) object is no longer associated with a [drag data store](#), abort these steps. Nothing happens.
2. If the [drag data store](#)'s mode is not the [read/write mode](#), abort these steps. Nothing happens.
3. Let *format* be the first argument, [converted to ASCII lowercase](#).
4. If *format* equals "text", change it to "text/plain".
If *format* equals "url", change it to "text/uri-list".
5. Remove the item in the [drag data store item list](#) whose **kind** is *Plain Unicode string* and whose [type string](#) is equal to *format*, if there is one.
6. Add an item to the [drag data store item list](#) whose **kind** is *Plain Unicode string*, whose [type string](#) is equal to *format*, and whose data is the string given by the method's second argument.

The **clearData()** method must run the following steps:

1. If the [DataTransfer](#) object is no longer associated with a [drag data store](#), abort these steps. Nothing happens.
2. If the [drag data store](#)'s mode is not the [read/write mode](#), abort these steps. Nothing happens.

3. If the method was called with no arguments, remove each item in the [drag data store item list](#) whose [kind](#) is *Plain Unicode string*, and abort these steps.
4. Let *format* be the first argument, [converted to ASCII lowercase](#).
5. If *format* equals "text", change it to "text/plain".
If *format* equals "url", change it to "text/uri-list".
6. Remove the item in the [drag data store item list](#) whose [kind](#) is *Plain Unicode string* and whose [type string](#) is equal to *format*, if there is one.

NOTE:

The [clearData\(\)](#) method does not affect whether any files were included in the drag, so the [types](#) attribute's list might still not be empty after calling [clearData\(\)](#) (it would still contain the "Files" string if any files were included in the drag).

The **files** attribute must return a [live](#) `FileList` sequence consisting of `File` objects representing the files found by the following steps. Furthermore, for a given `FileList` object and a given underlying file, the same `File` object must be used each time.

1. Start with an empty list *L*.
2. If the [DataTransfer](#) object is no longer associated with a [drag data store](#), the `FileList` is empty. Abort these steps; return the empty list *L*.
3. If the [drag data store](#)'s mode is the [protected mode](#), abort these steps; return the empty list *L*.
4. For each item in the [drag data store item list](#) whose [kind](#) is *File*, add the item's data (the file, in particular its name and contents, as well as its [type](#)) to the list *L*.
5. The files found by these steps are those in the list *L*.

NOTE:

This version of the API does not expose the types of the files during the drag.

5.7.3.1. The `DataTransferItemList` interface

Each [DataTransfer](#) object is associated with a [DataTransferItemList](#) object.

```
interface DataTransferItemList {
  readonly attribute unsigned long length;
  getter DataTransferItem (unsigned long index);
  DataTransferItem? add(DOMString data, DOMString type);
  DataTransferItem? add(File data);
  void remove(unsigned long index);
  void clear();
};
```

This definition is non-normative. Implementation requirements are given below this definition.

`items`.length

Returns the number of items in the [drag data store](#).

`items`[*index*]

Returns the [DataTransferItem](#) object representing the *index*th entry in the [drag data store](#).

`items`.remove(*index*)

Removes the *index*th entry in the [drag data store](#).

`items`.clear()

Removes all the entries in the [drag data store](#).

```

items . add( data )
items . add( data , type )

```

Adds a new entry for the given data to the [drag data store](#). If the data is plain text then a *type* string has to be provided also.

While the [DataTransferItemList](#) object's [DataTransfer](#) object is associated with a [drag data store](#), the [DataTransferItemList](#) object's *mode* is the same as the [drag data store mode](#). When the [DataTransferItemList](#) object's [DataTransfer](#) object is *not* associated with a [drag data store](#), the [DataTransferItemList](#) object's *mode* is the *disabled mode*. The [drag data store](#) referenced in this section (which is used only when the [DataTransferItemList](#) object is not in the *disabled mode*) is the [drag data store](#) with which the [DataTransferItemList](#) object's [DataTransfer](#) object is associated.

The **length** attribute must return zero if the object is in the *disabled mode*; otherwise it must return the number of items in the [drag data store item list](#).

When a [DataTransferItemList](#) object is not in the *disabled mode*, its [supported property indices](#) are the numbers in the range 0 .. *n*-1, where *n* is the number of items in the [drag data store item list](#).

To determine the value of an indexed property *i* of a [DataTransferItemList](#) object, the user agent must return a [DataTransferItem](#) object representing the *i*th item in the [drag data store](#). The same object must be returned each time a particular item is obtained from this [DataTransferItemList](#) object. The [DataTransferItem](#) object must be associated with the same [DataTransfer](#) object as the [DataTransferItemList](#) object when it is first created.

The **add()** method must run the following steps:

1. If the [DataTransferItemList](#) object is not in the *read/write mode*, return null and abort these steps.
2. Jump to the appropriate set of steps from the following list:

↪ **If the first argument to the method is a string**

If there is already an item in the [drag data store item list](#) whose *kind* is *Plain Unicode string* and whose *type string* is equal to the value of the method's second argument, [converted to ASCII lowercase](#), then throw a [NotSupportedError](#) exception and abort these steps.

Otherwise, add an item to the [drag data store item list](#) whose *kind* is *Plain Unicode string*, whose *type string* is equal to the value of the method's second argument, [converted to ASCII lowercase](#), and whose data is the string given by the method's first argument.

↪ **If the first argument to the method is a File**

Add an item to the [drag data store item list](#) whose *kind* is *File*, whose *type string* is the *type* of the *File*, [converted to ASCII lowercase](#), and whose data is the same as the *File*'s data.

3. **Determine the value of the indexed property** corresponding to the newly added item, and return that value (a newly created [DataTransferItem](#) object).

The **remove()** method, when invoked with the argument *i*, must run these steps:

1. If the [DataTransferItemList](#) object is not in the *read/write mode*, throw an *InvalidStateError* exception and abort these steps.
2. Remove the *i*th item from the [drag data store](#).

The **clear()** method, if the [DataTransferItemList](#) object is in the *read/write mode*, must remove all the items from the [drag data store](#). Otherwise, it must do nothing.

5.7.3.2. The [DataTransferItem](#) interface

Each [DataTransferItem](#) object is associated with a [DataTransfer](#) object.

```

interface DataTransferItem {
  readonly attribute DOMString kind;
  readonly attribute DOMString type;
  void getAsString(FunctionStringCallback? _callback);
  File? getAsFile();
};

callback FunctionStringCallback = void (DOMString data);

```

This definition is non-normative. Implementation requirements are given below this definition.

item . **kind**

Returns [the drag data item kind](#), one of: "string", "file".

item . **type**

Returns [the drag data item type string](#).

item . **getAsString**(*callback*)

Invokes the callback with the string data as the argument, if [the drag data item kind](#) is *Plain Unicode string*.

file = ***item*** . **getAsFile**()

Returns a File object, if [the drag data item kind](#) is *File*.

While the [DataTransferItem](#) object's [DataTransfer](#) object is associated with a [drag data store](#) and that [drag data store](#)'s [drag data store item list](#) still contains the item that the [DataTransferItem](#) object represents, the [DataTransferItem](#) object's *mode* is the same as the [drag data store mode](#). When the [DataTransferItem](#) object's [DataTransfer](#) object is *not* associated with a [drag data store](#), or if the item that the [DataTransferItem](#) object represents has been removed from the relevant [drag data store item list](#), the [DataTransferItem](#) object's *mode* is the *disabled mode*. The [drag data store](#) referenced in this section (which is used only when the [DataTransferItem](#) object is not in the *disabled mode*) is the [drag data store](#) with which the [DataTransferItem](#) object's [DataTransfer](#) object is associated.

The **kind** attribute must return the empty string if the [DataTransferItem](#) object is in the *disabled mode*; otherwise it must return the string given in the cell from the second column of the following table from the row whose cell in the first column contains [the drag data item kind](#) of the item represented by the [DataTransferItem](#) object:

Kind	String
<i>Plain Unicode string</i>	"string"
<i>File</i>	"file"

The **type** attribute must return the empty string if the [DataTransferItem](#) object is in the *disabled mode*; otherwise it must return [the drag data item type string](#) of the item represented by the [DataTransferItem](#) object.

The **getAsString**(*callback*) method must run the following steps:

1. If the *callback* is null, abort these steps.
2. If the [DataTransferItem](#) object is not in the *read/write mode* or the *read-only mode*, abort these steps. The callback is never invoked.
3. If [the drag data item kind](#) is not *Plain Unicode string*, abort these steps. The callback is never invoked.
4. Otherwise, [queue a task](#) to invoke *callback*, passing the actual data of the item represented by the [DataTransferItem](#) object as the argument.

The **getAsFile**() method must run the following steps:

1. If the [DataTransferItem](#) object is not in the *read/write mode* or the *read-only mode*, return null and abort these steps.
2. If [the drag data item kind](#) is not *File*, then return null and abort these steps.
3. Return a new File object representing the actual data of the item represented by the [DataTransferItem](#) object.

§ 5.7.4. The DragEvent interface

The drag-and-drop processing model involves several events. They all use the [DragEvent](#) interface.

```
[Constructor(DOMString type, optional DragEventInit eventInitDict)]
interface DragEvent : MouseEvent {
  readonly attribute DataTransfer? dataTransfer;
};

dictionary DragEventInit : MouseEventInit {
  DataTransfer? dataTransfer = null;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

event . **dataTransfer**

Returns the [DataTransfer](#) object for the event.

NOTE:

Although, for consistency with other event interfaces, the [DragEvent](#) interface has a constructor, it is not particularly useful. In particular, there's no way to create a useful [DataTransfer](#) object from script, as [DataTransfer](#) objects have a processing and security model that is coordinated by the browser during drag-and-drops.

The **dataTransfer** attribute of the [DragEvent](#) interface must return the value it was initialized to. It represents the context information for the event.

When a user agent is required to **fire a DND event** named *e* at an element, using a particular [drag data store](#), and optionally with a specific *related target*, the user agent must run the following steps:

1. If no specific *related target* was provided, set *related target* to null.
2. Let *window* be the Window object of the Document object of the specified target element.
3. If *e* is dragstart, set the [drag data store mode](#) to the [read/write mode](#).
If *e* is drop, set the [drag data store mode](#) to the [read-only mode](#).
4. Let *dataTransfer* be a newly created [DataTransfer](#) object associated with the given [drag data store](#).
5. Set the [effectAllowed](#) attribute to the [drag data store](#)'s [drag data store allowed effects state](#).
6. Set the [dropEffect](#) attribute to "none" if *e* is dragstart, drag, dragexit, or dragleave; to the value corresponding to the [current drag operation](#) if *e* is drop or dragend; and to a value based on the [effectAllowed](#) attribute's value and the drag-and-drop source, as given by the following table, otherwise (i.e., if *e* is dragenter or dragover):

effectAllowed	dropEffect
"none"	"none"
"copy"	"copy"
"copyLink"	"copy", or, if appropriate , "link"
"copyMove"	"copy", or, if appropriate , "move"
"all"	"copy", or, if appropriate , either "link" or "move"
"link"	"link"
"linkMove"	"link", or, if appropriate , "move"
"move"	"move"
"uninitialized" when what is being dragged is a selection from a text field	"move", or, if appropriate , either "copy" or "link"

effectAllowed	dropEffect
"uninitialized" when what is being dragged is a selection	"copy", or, if appropriate , either "link" or "move"
"uninitialized" when what is being dragged is an <a> element with an href attribute	"link", or, if appropriate , either "copy" or "move"
Any other case	"copy", or, if appropriate , either "link" or "move"

Where the table above provides **possibly appropriate alternatives**, user agents may instead use the listed alternative values if platform conventions dictate that the user has requested those alternate effects.

EXAMPLE 599

For example, Windows platform conventions are such that dragging while holding the "alt" key indicates a preference for linking the data, rather than moving or copying it. Therefore, on a Windows system, if "link" is an option according to the table above while the "alt" key is depressed, the user agent could select that instead of "copy" or "move".

7. Create a [trusted DragEvent](#) object and initialize it to have the given name `e`, to bubble, to be cancelable unless `e` is `dragexit`, `dragleave`, or `dragend`, and to have the view attribute initialized to `window`, the detail attribute initialized to zero, the mouse and key attributes initialized according to the state of the input devices as they would be for user interaction events, the `relatedTarget` attribute initialized to *related target*, and the [dataTransfer](#) attribute initialized to `dataTransfer`, the [DataTransfer](#) object created above.

If there is no relevant pointing device, the object must have its `screenX`, `screenY`, `clientX`, `clientY`, and `button` attributes set to 0.

8. [Dispatch](#) the newly created [DragEvent](#) object at the specified target element.
9. Set the [drag data store allowed effects state](#) to the current value of `dataTransfer`'s [effectAllowed](#) attribute. (It can only have changed value if `e` is `dragstart`.)
10. Set the [drag data store mode](#) back to the [protected mode](#) if it was changed in the first step.
11. Break the association between `dataTransfer` and the [drag data store](#).

§ 5.7.5. Drag-and-drop processing model

When the user attempts to begin a drag operation, the user agent must run the following steps. User agents must act as if these steps were run even if the drag actually started in another document or application and the user agent was not aware that the drag was occurring until it intersected with a document under the user agent's purview.

1. Determine what is being dragged, as follows:

If the drag operation was invoked on a selection, then it is the selection that is being dragged.

Otherwise, if the drag operation was invoked on a [Document](#), it is the first element, going up the ancestor chain, starting at the node that the user tried to drag, that has the IDL attribute [draggable](#) set to true. If there is no such element, then nothing is being dragged; abort these steps, the drag-and-drop operation is never started.

Otherwise, the drag operation was invoked outside the user agent's purview. What is being dragged is defined by the document or application where the drag was started.

NOTE:

[](#) elements and [<a>](#) elements with an [href](#) attribute have their [draggable](#) attribute set to true by default.

2. [Create a drag data store](#). All the DND events fired subsequently by the steps in this section must use this [drag data store](#).
3. Establish which DOM node is the **source node**, as follows:

If it is a selection that is being dragged, then the [source node](#) is the Text node that the user started the drag on (typically the Text node that the user originally clicked). If the user did not specify a particular node, for example if the user just told the user agent to begin a drag of "the selection", then the [source node](#) is the first Text node containing a part of the selection.

Otherwise, if it is an element that is being dragged, then the [source node](#) is the element that is being dragged.

Otherwise, the [source node](#) is part of another document or application. When this specification requires that an event be dispatched at the [source node](#) in this case, the user agent must instead follow the platform-specific conventions relevant to that situation.

NOTE:

Multiple events are fired on the [source node](#) during the course of the drag-and-drop operation.

4. Determine the **list of dragged nodes**, as follows:

If it is a selection that is being dragged, then the [list of dragged nodes](#) contains, in [tree order](#), every node that is partially or completely included in the selection (including all their ancestors).

Otherwise, the [list of dragged nodes](#) contains only the [source node](#), if any.

5. If it is a selection that is being dragged, then add an item to the [drag data store item list](#), with its properties set as follows:

The drag data item type string

"text/plain"

The drag data item kind

Plain Unicode string

The actual data

The text of the selection

Otherwise, if any files are being dragged, then add one item per file to the [drag data store item list](#), with their properties set as follows:

The drag data item type string

The MIME type of the file, if known, or "application/octet-stream" otherwise.

The drag data item kind

File

The actual data

The file's contents and name.

NOTE:

Dragging files can currently only happen from outside a [browsing context](#), for example from a file system manager application.

If the drag initiated outside of the application, the user agent must add items to the [drag data store item list](#) as appropriate for the data being dragged, honoring platform conventions where appropriate; however, if the platform conventions do not use [MIME types](#) to label dragged data, the user agent must make a best-effort attempt to map the types to MIME types, and, in any case, all the [drag data item type strings](#) must be [converted to ASCII lowercase](#).

User agents may also add one or more items representing the selection or dragged element(s) in other forms, e.g., as HTML.

6. If the [list of dragged nodes](#) is not empty, then extract the microdata from those nodes into a JSON form, and add one item to the [drag data store item list](#), with its properties set as follows:

The drag data item type string

application/microdata+json

The drag data item kind*Plain Unicode string***The actual data**

The resulting JSON string.

7. Run the following substeps:

1. Let *urls* be an empty list of [absolute URLs](#).2. For each *node* in the [list of dragged nodes](#):**If the node is an `<a>` element with an `href` attribute**Add to *urls* the result of [parsing](#) the element's `href` content attribute relative to the element.**If the node is an `<img>` element with a `src` attribute**Add to *urls* the result of [parsing](#) the element's `src` content attribute relative to the element.3. If *urls* is still empty, abort these substeps.4. Let *url string* be the result of concatenating the strings in *urls*, in the order they were added, separated by a U+000D CARRIAGE RETURN U+000A LINE FEED character pair (CRLF).5. Add one item to the [drag data store item list](#), with its properties set as follows:**The drag data item type string**

text/uri-list

The drag data item kind*Plain Unicode string***The actual data***url string*

8. Update the [drag data store default feedback](#) as appropriate for the user agent (if the user is dragging the selection, then the selection would likely be the basis for this feedback; if the user is dragging an element, then that element's rendering would be used; if the drag began outside the user agent, then the platform conventions for determining the drag feedback should be used).

9. [Fire a DND event](#) named `dragstart` at the [source node](#).

If the event is canceled, then the drag-and-drop operation should not occur; abort these steps.

NOTE:

Since events with no event listeners registered are, almost by definition, never canceled, drag-and-drop is always available to the user if the author does not specifically prevent it.

10. [Initiate the drag-and-drop operation](#) in a manner consistent with platform conventions, and as described below.

The drag-and-drop feedback must be generated from the first of the following sources that is available:

1. The [drag data store bitmap](#), if any. In this case, the [drag data store hot spot coordinate](#) should be used as hints for where to put the cursor relative to the resulting image. The values are expressed as distances in CSS pixels from the left side and from the top side of the image respectively. [\[CSS-2015\]](#)
2. The [drag data store default feedback](#).

From the moment that the user agent is to **initiate the drag-and-drop operation**, until the end of the drag-and-drop operation, device input events (e.g., mouse and keyboard events) must be suppressed.

During the drag operation, the element directly indicated by the user as the drop target is called the **immediate user selection**. (Only elements can be selected by the user; other nodes must not be made available as drop targets.) However, the [immediate user selection](#) is not necessarily the **current target element**, which is the element currently selected for the drop part of the drag-and-drop operation.

The [immediate user selection](#) changes as the user selects different elements (either by pointing at them with a pointing device, or by selecting them in some other way). The [current target element](#) changes when the [immediate user selection](#) changes, based on the results of event listeners in the document, as described below.

Both the [current target element](#) and the [immediate user selection](#) can be null, which means no target element is selected. They can also both be elements in other (DOM-based) documents, or other (non-Web) programs altogether. (For example, a user could drag text to a word-processor.) The [current target element](#) is initially null.

In addition, there is also a **current drag operation**, which can take on the values "**none**", "**copy**", "**link**", and "**move**". Initially, it has the value "none". It is updated by the user agent as described in the steps below.

User agents must, as soon as the drag operation is [initiated](#) and every 350ms (± 200 ms) thereafter for as long as the drag operation is ongoing, [queue a task](#) to perform the following steps in sequence:

1. If the user agent is still performing the previous iteration of the sequence (if any) when the next iteration becomes due, abort these steps for this iteration (effectively "skipping missed frames" of the drag-and-drop operation).
2. [Fire a DND event](#) named drag at the [source node](#). If this event is canceled, the user agent must set the [current drag operation](#) to "none" (no drag operation).
3. If the drag event was not canceled and the user has not ended the drag-and-drop operation, check the state of the drag-and-drop operation, as follows:
 1. If the user is indicating a different [immediate user selection](#) than during the last iteration (or if this is the first iteration), and if this [immediate user selection](#) is not the same as the [current target element](#), then [fire a DND event](#) named dragexit at the [current target element](#), and then update the [current target element](#) as follows:

↪ If the new [immediate user selection](#) is null

Set the [current target element](#) to null also.

↪ If the new [immediate user selection](#) is in a non-DOM document or application

Set the [current target element](#) to the [immediate user selection](#).

↪ Otherwise

[Fire a DND event](#) named dragenter at the [immediate user selection](#).

If the event is canceled, then set the [current target element](#) to the [immediate user selection](#).

Otherwise, run the appropriate step from the following list:

↪ If the [immediate user selection](#) is a text field (e.g., `<textarea>`, or an `<input>` element whose [type](#) attribute is in the [Text](#) state) or an [editing host](#) or [editable](#) element, and the [drag data store item list](#) has an item with the [drag data item type string](#) "text/plain" and the [drag data item kind](#) *Plain Unicode string*

Set the [current target element](#) to the [immediate user selection](#) anyway.

↪ If the [immediate user selection](#) is an element with a [dropzone](#) attribute that [matches the drag data store](#)

Set the [current target element](#) to the [immediate user selection](#) anyway.

↪ If the [immediate user selection](#) is an element that itself has an ancestor element with a [dropzone](#) attribute that [matches the drag data store](#)

Let *new target* be the nearest (deepest) such ancestor element.

If the [immediate user selection](#) is *new target*, then leave the [current target element](#) unchanged.

Otherwise, [fire a DND event](#) named dragenter at *new target*, with the current [current target element](#) as the specific [related target](#). Then, set the [current target element](#) to *new target*, regardless of whether that event was canceled or not.

↪ If the [immediate user selection](#) is the [body element](#)

Leave the [current target element](#) unchanged.

↪ Otherwise

[Fire a DND event](#) named dragenter at the [body element](#), if there is one, or at the Document object, if not. Then, set the [current target element](#) to the [body element](#), regardless of whether

that event was canceled or not.

- If the previous step caused the [current target element](#) to change, and if the previous target element was not null or a part of a non-DOM document, then [fire a DND event](#) named `dragleave` at the previous target element, with the new [current target element](#) as the specific *related target*.
- If the [current target element](#) is a DOM element, then [fire a DND event](#) named `dragover` at this [current target element](#).

If the `dragover` event is not canceled, run the appropriate step from the following list:

- ↪ If the [current target element](#) is a text field (e.g., `<textarea>`, or an `<input>` element whose `type` attribute is in the [Text](#) state) or an [editing host](#) or [editable](#) element, and the [drag data store item list](#) has an item with [the drag data item type string](#) `"text/plain"` and [the drag data item kind](#) *Plain Unicode string*

Set the [current drag operation](#) to either `"copy"` or `"move"`, as appropriate given the platform conventions.

- ↪ If the [current target element](#) is an element with a [dropzone](#) attribute that [matches](#) the [drag data store](#) and [specifies an operation](#)

Set the [current drag operation](#) to the operation [specified](#) by the [dropzone](#) attribute of the [current target element](#).

- ↪ If the [current target element](#) is an element with a [dropzone](#) attribute that [matches](#) the [drag data store](#) and does not [specify an operation](#)

Set the [current drag operation](#) to `"copy"`.

- ↪ Otherwise

Reset the [current drag operation](#) to `"none"`.

Otherwise (if the `dragover` event *is* canceled), set the [current drag operation](#) based on the values of the [effectAllowed](#) and [dropEffect](#) attributes of the [DragEvent](#) object's [dataTransfer](#) object as they stood after the event [dispatch](#) finished, as per the following table:

effectAllowed	dropEffect	Drag operation
"uninitialized", "copy", "copyLink", "copyMove", or "all"	"copy"	"copy"
"uninitialized", "link", "copyLink", "linkMove", or "all"	"link"	"link"
"uninitialized", "move", "copyMove", "linkMove", or "all"	"move"	"move"
Any other case		"none"

- Otherwise, if the [current target element](#) is not a DOM element, use platform-specific mechanisms to determine what drag operation is being performed (none, copy, link, or move), and set the [current drag operation](#) accordingly.
- Update the drag feedback (e.g., the mouse cursor) to match the [current drag operation](#), as follows:

Drag operation	Feedback
"copy"	Data will be copied if dropped here.
"link"	Data will be linked if dropped here.
"move"	Data will be moved if dropped here.
"none"	No operation allowed, dropping here will cancel the drag-and-drop operation.

- Otherwise, if the user ended the drag-and-drop operation (e.g., by releasing the mouse button in a mouse-driven drag-and-drop interface), or if the drag event was canceled, then this will be the last iteration. Run the following steps, then stop the drag-and-drop operation:

- If the [current drag operation](#) is `"none"` (no drag operation), or, if the user ended the drag-and-drop operation by canceling it (e.g., by hitting the `Escape` key), or if the [current target element](#) is null, then the drag operation failed. Run these substeps:

- Let `dropped` be false.

- If the [current target element](#) is a DOM element, [fire a DND event](#) named `dragleave` at it; otherwise, if it is

not null, use platform-specific conventions for drag cancelation.

3. Set the [current drag operation](#) to "none".

Otherwise, the drag operation might be a success; run these substeps:

1. Let *dropped* be true.
2. If the [current target element](#) is a DOM element, [fire a DND event](#) named drop at it; otherwise, use platform-specific conventions for indicating a drop.
3. If the event is canceled, set the [current drag operation](#) to the value of the [dropEffect](#) attribute of the [DragEvent](#) object's [dataTransfer](#) object as it stood after the event [dispatch](#) finished.

Otherwise, the event is not canceled; perform the event's default action, which depends on the exact target as follows:

- ↪ If the [current target element](#) is a text field (e.g., `<textarea>`, or an `<input>` element whose [type](#) attribute is in the [Text](#) state) or an [editing host](#) or [editable](#) element, and the [drag data store item list](#) has an item with the [drag data item type string](#) "text/plain" and the [drag data item kind](#) *Plain Unicode string*

Insert the actual data of the first item in the [drag data store item list](#) to have a [drag data item type string](#) of "text/plain" and a [drag data item kind](#) that is *Plain Unicode string* into the text field or [editing host](#) or [editable](#) element in a manner consistent with platform-specific conventions (e.g., inserting it at the current mouse cursor position, or inserting it at the end of the field).

- ↪ Otherwise

Reset the [current drag operation](#) to "none".

2. [Fire a DND event](#) named dragend at the [source node](#).

3. Run the appropriate steps from the following list as the default action of the dragend event:

- ↪ If *dropped* is true, the [current target element](#) is a *text field* (see below), the [current drag operation](#) is "move", and the source of the drag-and-drop operation is a selection in the DOM that is entirely contained within an [editing host](#)

[Delete the selection](#).

- ↪ If *dropped* is true, the [current target element](#) is a *text field* (see below), the [current drag operation](#) is "move", and the source of the drag-and-drop operation is a selection in a text field

The user agent should delete the dragged selection from the relevant text field.

- ↪ If *dropped* is false or if the [current drag operation](#) is "none"

The drag was canceled. If the platform conventions dictate that this be represented to the user (e.g., by animating the dragged selection going back to the source of the drag-and-drop operation), then do so.

- ↪ Otherwise

The event has no default action.

For the purposes of this step, a *text field* is a `<textarea>` element or an `<input>` element whose [type](#) attribute is in one of the [Text](#), [Search](#), [Telephone](#), [URL](#), [E-mail](#), [Password](#), or [Number](#) states.

NOTE:

User agents are encouraged to consider how to react to drags near the edge of scrollable regions. For example, if a user drags a link to the bottom of the [viewport](#) on a long page, it might make sense to scroll the page so that the user can drop the link lower on the page.

NOTE:

This model is independent of which Document object the nodes involved are from; the events are fired as described above and the rest of the processing model runs as described above, irrespective of how many documents are involved in the operation.

§ 5.7.6. Events summary

This section is non-normative.

The following events are involved in the drag-and-drop model.

Event Name	Target	Cancelable?	Drag data store mode	dropEffect	Default Action
dragstart	Source node	✓ Cancelable	Read/write mode	"none"	Initiate the drag-and-drop operation
drag	Source node	✓ Cancelable	Protected mode	"none"	Continue the drag-and-drop operation
dragenter	Immediate user selection or the body element	✓ Cancelable	Protected mode	Based on effectAllowed value	Reject immediate user selection as potential target element
dragexit	Previous target element	—	Protected mode	"none"	None
dragleave	Previous target element	—	Protected mode	"none"	None
dragover	Current target element	✓ Cancelable	Protected mode	Based on effectAllowed value	Reset the current drag operation to "none"
drop	Current target element	✓ Cancelable	Read-only mode	Current drag operation	Varies
dragend	Source node	—	Protected mode	Current drag operation	Varies

Not shown in the above table: all these events bubble, and the [effectAllowed](#) attribute always has the value it had after the dragstart event, defaulting to "uninitialized" in the dragstart event.

§ 5.7.7. The draggable attribute

All [html elements](#) may have the **draggable** content attribute set. The [draggable](#) attribute is an [enumerated attribute](#). It has three states. The first state is *true* and it has the keyword `true`. The second state is *false* and it has the keyword `false`. The third state is *auto*; it has no keywords but it is the *missing value default*.

The *true* state means the element is draggable; the *false* state means that it is not. The *auto* state uses the default behavior of the user agent.

An element with a [draggable](#) attribute should also have a `title` attribute that names the element for the purpose of non-visual interactions.

This definition is non-normative. Implementation requirements are given below this definition.

***element* . [draggable](#) [= *value*]**

Returns true if the element is draggable; otherwise, returns false.

Can be set, to override the default and set the [draggable](#) content attribute.

The **draggable** IDL attribute, whose value depends on the content attribute's in the way described below, controls whether or not the element is draggable. Generally, only text selections are draggable, but elements whose [draggable](#) IDL attribute is true become draggable as well.

If an element's [draggable](#) content attribute has the state *true*, the [draggable](#) IDL attribute must return true.

Otherwise, if the element's [draggable](#) content attribute has the state *false*, the [draggable](#) IDL attribute must return false.

Otherwise, the element's [draggable](#) content attribute has the state *auto*. If the element is an `<img>` element, an `<object>` element that [represents](#) an image, or an `<a>` element with an [href](#) content attribute, the [draggable](#) IDL attribute must return

true; otherwise, the [draggable](#) IDL attribute must return false.

If the [draggable](#) IDL attribute is set to the value false, the [draggable](#) content attribute must be set to the literal value "false". If the [draggable](#) IDL attribute is set to the value true, the [draggable](#) content attribute must be set to the literal value "true".

§ 5.7.8. The dropzone attribute

All [html elements](#) may have the **dropzone** content attribute set. When specified, its value must be an [unordered set of unique space-separated tokens](#) that are [ASCII case-insensitive](#). The allowed values are the following:

copy

Indicates that dropping an accepted item on the element will result in a copy of the dragged data.

move

Indicates that dropping an accepted item on the element will result in the dragged data being moved to the new location.

link

Indicates that dropping an accepted item on the element will result in a link to the original data.

Any keyword with eight characters or more, beginning with an [ASCII case-insensitive](#) match for the string "string:"

Indicates that items with [the drag data item kind](#) *Plain Unicode string* and [the drag data item type string](#) set to a value that matches the remainder of the keyword are accepted.

Any keyword with six characters or more, beginning with an [ASCII case-insensitive](#) match for the string "file:"

Indicates that items with [the drag data item kind](#) *File* and [the drag data item type string](#) set to a value that matches the remainder of the keyword are accepted.

The [dropzone](#) content attribute's values must not have more than one of the three feedback values (copy, move, and link) specified. If none are specified, the copy value is implied.

An element with a [dropzone](#) attribute should also have a `title` attribute that names the element for the purpose of non-visual interactions.

A [dropzone](#) attribute **matches a drag data store** if the [dropzone processing steps](#) result in a match.

A [dropzone](#) attribute **specifies an operation** if the [dropzone processing steps](#) result in a specified operation. The specified operation is as given by those steps.

The [dropzone processing steps](#) are as follows. They either result in a match or not, and separate from this result either in a specified operation or not, as defined below.

1. Let `value` be the value of the [dropzone](#) attribute.
2. Let `keywords` be the result of [splitting value on spaces](#).
3. Let `matched` be false.
4. Let `operation` be unspecified.
5. For each value in `keywords`, if any, in the order that they were found in `value`, run the following steps.
 1. Let `keyword` be the keyword.
 2. If `keyword` is one of "copy", "move", or "link", then: run the following substeps:
 1. If `operation` is still unspecified, then let `operation` be the string given by `keyword`.
 2. Skip to the step labeled *end of keyword* below.
 3. If `keyword` does not contain a U+003A COLON character (:), or if the first such character in `keyword` is either the first character or the last character in the string, then skip to the step labeled *end of keyword* below.

4. Let *kind code* be the substring of *keyword* from the first character in the string to the last character in the string that is before the first U+003A COLON character (:) in the string, [converted to ASCII lowercase](#).

5. Jump to the appropriate step from the list below, based on the value of *kind code*:

↪ If *kind code* is the string "string"

Let *kind* be Plain Unicode string.

↪ If *kind code* is the string "file"

Let *kind* be File.

↪ Otherwise

Skip to the step labeled *end of keyword* below.

6. Let *type* be the substring of *keyword* from the first character after the first U+003A COLON character (:) in the string, to the last character in the string, [converted to ASCII lowercase](#).

7. If there exist any items in the [drag data store item list](#) whose [drag data item kind](#) is the kind given in *kind* and whose [drag data item type string](#) is *type*, then let *matched* be true.

8. *End of keyword*: Go on to the next keyword, if any, or the next step in the overall algorithm, if there are no more.

6. The algorithm results in a match if *matched* is true, and does not otherwise.

The algorithm results in a specified operation if *operation* is not unspecified. The specified operation, if one is specified, is the one given by *operation*.

The **dropzone** IDL attribute must [reflect](#) the content attribute of the same name.

The [supported tokens](#) for [dropzone](#) are the allowed values defined for the [dropzone](#) attribute that are supported by the user agent.

EXAMPLE 600

In this example, a `<div>` element is made into a drop target for image files using the [dropzone](#) attribute. Images dropped into the target are then displayed.

```
<div dropzone="copy file:image/png file:image/gif file:image/jpeg" ondrop="receive(event,
this)">
<p>Drop an image here to have it displayed.</p>
</div>
<script>
function receive(event, element) {
  var data = event.dataTransfer.items;
  for (var i = 0; i < data.length; i += 1) {
    if ((data[i].kind == 'file') && (data[i].type.match('^image/')) {
      var img = new Image();
      img.src = window.createObjectURL(data[i].getAsFile());
      element.appendChild(img);
    }
  }
}
</script>
```

§ 5.7.9. Security risks in the drag-and-drop model

User agents must not make the data added to the [DataTransfer](#) object during the `dragstart` event available to scripts until the drop event, because otherwise, if a user were to drag sensitive information from one document to a second document, crossing a hostile third document in the process, the hostile document could intercept the data.

For the same reason, user agents must consider a drop to be successful only if the user specifically ended the drag operation — if any scripts end the drag operation, it must be considered unsuccessful (canceled) and the drop event must not be fired.

User agents should take care to not start drag-and-drop operations in response to script actions. For example, in a mouse-and-window environment, if a script moves a window while the user has his mouse button depressed, the user agent would not consider that to start a drag. This is important because otherwise user agents could cause data to be dragged from sensitive sources and dropped into hostile documents without the user's consent.

User agents should filter potentially active (scripted) content (e.g., HTML) when it is dragged and when it is dropped, using a safelist of known-safe features. Similarly, [relative URLs](#) should be turned into absolute URLs to avoid references changing in unexpected ways. This specification does not specify how this is performed.

EXAMPLE 601

Consider a hostile page providing some content and getting the user to select and drag and drop (or indeed, copy and paste) that content to a victim page's [contenteditable](#) region. If the browser does not ensure that only safe content is dragged, potentially unsafe content such as scripts and event handlers in the selection, once dropped (or pasted) into the victim site, get the privileges of the victim site. This would thus enable a cross-site scripting attack.

§ 6. Loading Web pages

This section describes features that apply most directly to Web browsers. Having said that, except where specified otherwise, the requirements defined in this section *do* apply to all user agents, whether they are Web browsers or not.

§ 6.1. Browsing contexts

A **browsing context** is an environment in which [Document](#) objects are presented to the user.

NOTE:

A tab or window in a Web browser typically contains a [browsing context](#), as does an `<iframe>` or `<frame>`s in a `<frameset>`.

A [browsing context](#) has a corresponding [WindowProxy](#) object.

A [browsing context](#) has a [session history](#), which lists the [Document](#) objects that the [browsing context](#) has presented, is presenting, or will present. At any time, one [Document](#) in each [browsing context](#) is designated the **active document**. A [Document](#)'s [browsing context](#) is that [browsing context](#) whose [session history](#) contains the [Document](#), if any. (A [Document](#) created using an API such as `createDocument()` has no [browsing context](#).) Each [Document](#) in a [browsing context](#) is **associated** with a [Window](#) object.

NOTE:

In general, there is a 1-to-1 mapping from the [Window](#) object to the [Document](#) object. There are two exceptions. First, a [Window](#) can be reused for the presentation of a second [Document](#) in the same [browsing context](#), such that the mapping is then 1-to-2. This occurs when a [browsing context](#) is [navigated](#) from the initial `about:blank` [Document](#) to another, with [replacement enabled](#). Second, a [Document](#) can end up being reused for several [Window](#) objects when the `document.open()` method is used, such that the mapping is then many-to-1.

NOTE:

A [Document](#) does not necessarily have a [browsing context](#) associated with it. In particular, data mining tools are likely to never instantiate browsing contexts.



A [browsing context](#) can have a **creator browsing context**, the [browsing context](#) that was responsible for its creation. If a [browsing context](#) has a [parent browsing context](#), then that is its [creator browsing context](#). Otherwise, if the [browsing context](#) has an [opener browsing context](#), then *that* is its [creator browsing context](#). Otherwise, the [browsing context](#) has no [creator browsing context](#).

If a [browsing context](#) *A* has a [creator browsing context](#), then the **creator origin**, **creator URL**, and **creator base URL** are

the [origin](#), [URL](#), and [base URL](#), respectively, of the [Document](#) that was the [active document](#) of that [creator browsing context](#) at the time *A* was created.

To create a new browsing context:

1. Call the JavaScript [InitializeHostDefinedRealm\(\)](#) abstract operation with the following customizations:
 - For the global object, create a new [Window](#) object *window*.
 - For the global *this* value, create a new [WindowProxy](#) object *windowProxy*, whose [\[\[Window\]\]](#) internal slot value is *window*.
- NOTE:**
The internal slot value is updated when navigations occur.
- Let *realm execution context* be the created [JavaScript execution context](#).
2. Set the new [browsing context](#)'s associated [WindowProxy](#) to *windowProxy*.
3. Let *document* be a new [Document](#), whose [URL](#) is [about:blank](#), which is marked as being an [HTML document](#), whose [character encoding](#) is UTF-8, and which is both [ready for post-load tasks](#) and [completely loaded](#) immediately.
4. Set the [origin](#) of *document*:
 - If the new [browsing context](#) has a [creator browsing context](#), then the [origin](#) of *document* is the [creator origin](#).
 - Otherwise, the [origin](#) of *document* is a unique [opaque origin](#) assigned when the new [browsing context](#) is created.
5. If the new [browsing context](#) has a [creator browsing context](#), then set *document*'s [referrer](#) to the [creator URL](#).
6. Ensure that *document* has a single child [<html>](#) node, which itself has two empty child nodes: a [<head>](#) element, and a [<body>](#) element.
7. [Implement the sandboxing](#) for *document*.
8. Add *document* to the new [browsing context](#)'s [session history](#).
9. Set *window*'s associated [Document](#) to *document*.
10. [Set up a browsing context environment settings object](#) with *realm execution context*.

§ 6.1.1. Nested browsing contexts

Certain elements (for example, [<iframe>](#) elements) can instantiate further [browsing contexts](#). These are called nested browsing contexts. If a browsing context *P* has a [Document](#) *D* with an element *E* that nests another browsing context *C* inside it, then *C* is said to be **nested through** *D*, and *E* is said to be the **browsing context container** of *C*. If the [browsing context container](#) element *E* is [in the Document](#) *D*, then *P* is said to be the **parent browsing context** of *C* and *C* is said to be a **child browsing context** of *P*. Otherwise, the [nested browsing context](#) *C* has no [parent browsing context](#).

A browsing context *A* is said to be an **ancestor** of a browsing context *B* if there exists a browsing context *A'* that is a [child browsing context](#) of *A* and that is itself an [ancestor](#) of *B*, or if the browsing context *A* is the [parent browsing context](#) of *B*.

A browsing context that is not a [nested browsing context](#) has no [parent browsing context](#), and is the **top-level browsing context** of all the browsing contexts for which it is an [ancestor browsing context](#).

The transitive closure of [parent browsing contexts](#) for a [nested browsing context](#) gives the list of [ancestor browsing contexts](#).

The **list of the descendant browsing contexts** of a [Document](#) *d* is the (ordered) list returned by the following algorithm:

1. Let *list* be an empty list.
2. For each [child browsing context](#) of *d* that is [nested through](#) an element that is [in the Document](#) *d*, in the [tree order](#) of the elements nesting those [browsing contexts](#), run these substeps:
 1. Append that [child browsing context](#) to the list *list*.

2. Append the [list of the descendant browsing contexts](#) of the [active document](#) of that [child browsing context](#) to the list *list*.

3. Return the constructed *list*.

A [Document](#) is said to be **fully active** when it has a [browsing context](#) and it is the [active document](#) of that [browsing context](#), and either its browsing context is a [top-level browsing context](#), or it has a [parent browsing context](#) and the [Document](#) through which it is nested is itself [fully active](#).

Because they are nested through an element, [child browsing contexts](#) are always tied to a specific [Document](#) in their [parent browsing context](#). User agents must not allow the user to interact with [child browsing contexts](#) of elements that are in [Documents](#) that are not themselves [fully active](#).

A [nested browsing context](#) can be put into a **delaying load events mode**. This is used when it is [navigated](#), to [delay the load event](#) of the [browsing context container](#) before the new [Document](#) is created.

The **document family** of a [browsing context](#) consists of the union of all the [Document](#) objects in that [browsing context](#)'s [session history](#) and the [document families](#) of all those [Document](#) objects. The [document family](#) of a [Document](#) object consists of the union of all the [document families](#) of the [browsing contexts](#) that are [nested through](#) the [Document](#) object.

§ 6.1.1.1. Navigating nested browsing contexts in the DOM

This definition is non-normative. Implementation requirements are given below this definition.

`window` . `top`

Returns the [WindowProxy](#) for the [top-level browsing context](#).

`window` . `parent`

Returns the [WindowProxy](#) for the [parent browsing context](#).

`window` . `frameElement`

Returns the [Element](#) for the [browsing context container](#).

Returns null if there isn't one, and in cross-origin situations.

The **top** IDL attribute on the [Window](#) object of a [Document](#) in a [browsing context](#) *b* must return the [WindowProxy](#) object of its [top-level browsing context](#) (which would be its own [WindowProxy](#) object if it was a [top-level browsing context](#) itself), if it has one, or its own [WindowProxy](#) object otherwise (e.g., if it was a detached [nested browsing context](#)).

The **parent** IDL attribute on the [Window](#) object of a [Document](#) that has a [browsing context](#) *b* must return the [WindowProxy](#) object of the [parent browsing context](#), if there is one (i.e., if *b* is a [child browsing context](#)), or the [WindowProxy](#) object of the [browsing context](#) *b* itself, otherwise (i.e., if it is a [top-level browsing context](#) or a detached [nested browsing context](#)).

The **frameElement** IDL attribute, on getting, must run the following algorithm:

1. Let *d* be the [Window](#) object's [associated Document](#).
2. Let *context* be *d*'s [browsing context](#).
3. If *context* is not a [nested browsing context](#), return null and abort these steps.
4. Let *container* be *context*'s [browsing context container](#).
5. If *container*'s [node document](#)'s [origin](#) is not [same origin-domain](#) with the [entry settings object](#)'s [origin](#), then return null and abort these steps.
6. Return *container*.

§ 6.1.2. Auxiliary browsing contexts

It is possible to create new browsing contexts that are related to a [top-level browsing context](#) without being nested through an element. Such browsing contexts are called [auxiliary browsing contexts](#). Auxiliary browsing contexts are always [top-](#)

[level browsing contexts](#).

An [auxiliary browsing context](#) has an **opener browsing context**, which is the [browsing context](#) from which the [auxiliary browsing context](#) was created.

§ 6.1.2.1. Navigating auxiliary browsing contexts in the DOM

The **opener** IDL attribute on the `Window` object, on getting, must return the `WindowProxy` object of the [browsing context](#) from which the current [browsing context](#) was created (its [opener browsing context](#)), if there is one, if it is still available, and if the current [browsing context](#) has not *disowned its opener*; otherwise, it must return null. On setting, if the new value is null then the current [browsing context](#) must **disown its opener**; if the new value is anything else then the user agent must call the `[[DefineOwnProperty]]` internal method of the `Window` object, passing the property name "opener" as the property key, and the Property Descriptor { `[[Value]]`: *value*, `[[Writable]]`: true, `[[Enumerable]]`: true, `[[Configurable]]`: true } as the property descriptor, where *value* is the new value.

§ 6.1.3. Security

A [browsing context](#) *A* is **familiar with** a second [browsing context](#) *B* if one of the following conditions is true:

- Either the [origin](#) of the [active document](#) of *A* is the same as the [origin](#) of the [active document](#) of *B*, or
- The browsing context *A* is a [nested browsing context](#) with a [top-level browsing context](#), and its [top-level browsing context](#) is *B*, or
- The browsing context *B* is an [auxiliary browsing context](#) and *A* is familiar with *B*'s [opener browsing context](#), or
- The browsing context *B* is not a [top-level browsing context](#), but there exists an [ancestor browsing context](#) of *B* whose [active document](#) has the [same origin](#) as the [active document](#) of *A* (possibly in fact being *A* itself).



A [browsing context](#) *A* is **allowed to navigate** a second [browsing context](#) *B* if the following algorithm terminates positively:

1. If *A* is not the same [browsing context](#) as *B*, and *A* is not one of the [ancestor browsing contexts](#) of *B*, and *B* is not a [top-level browsing context](#), and *A*'s [active document](#)'s [active sandboxing flag set](#) has its [sandboxed navigation browsing context flag set](#), then abort these steps negatively.
2. Otherwise, if *B* is a [top-level browsing context](#), and is one of the [ancestor browsing contexts](#) of *A*, and *A*'s [active document](#)'s [active sandboxing flag set](#) has its [sandboxed top-level navigation browsing context flag set](#), then abort these steps negatively.
3. Otherwise, if *B* is a [top-level browsing context](#), and is neither *A* nor one of the [ancestor browsing contexts](#) of *A*, and *A*'s [Document's active sandboxing flag set](#) has its [sandboxed navigation browsing context flag set](#), and *A* is not the [one permitted sandboxed navigator](#) of *B*, then abort these steps negatively.
4. Otherwise, terminate positively!



An element has a **browsing context scope origin** if its [Document's browsing context](#) is a [top-level browsing context](#) or if all of its [Document's ancestor browsing contexts](#) all have [active documents](#) whose [origin](#) are the [same origin](#) as the element's [node document's origin](#). If an element has a [browsing context scope origin](#), then its value is the [origin](#) of the element's [node document](#).

§ 6.1.4. Groupings of browsing contexts

Each [browsing context](#) is defined as having a list of one or more **directly reachable browsing contexts**. These are:

- The [browsing context](#) itself.
- All the [browsing context](#)'s [child browsing contexts](#).
- The [browsing context](#)'s [parent browsing context](#).
- All the [browsing contexts](#) that have the [browsing context](#) as their [opener browsing context](#).
- The [browsing context](#)'s [opener browsing context](#).

The transitive closure of all the [browsing contexts](#) that are [directly reachable browsing contexts](#) forms a **unit of related browsing contexts**.

Each [unit of related browsing contexts](#) is then further divided into the smallest number of groups such that every member of each group has an [active document](#) with an [origin](#) that, through appropriate manipulation of the `document.domain` attribute, could be made to be [same origin-domain](#) with other members of the group, but could not be made the same as members of any other group. Each such group is a **unit of related similar-origin browsing contexts**.

NOTE:

There is also at most one [event loop](#) per [unit of related similar-origin browsing contexts](#) (though several [units of related similar-origin browsing contexts](#) can have a shared [event loop](#)).

§ 6.1.5. Browsing context names

Browsing contexts can have a **browsing context name**. By default, a browsing context has no name (its name is not set).

A **valid browsing context name** is any string with at least one character that does not start with a U+005F LOW LINE character. (Names starting with an underscore are reserved for special keywords.)

A **valid browsing context name or keyword** is any string that is either a [valid browsing context name](#) or that is an [ASCII case-insensitive](#) match for one of: `_blank`, `_self`, `_parent`, or `_top`.

These values have different meanings based on whether the page is sandboxed or not, as summarized in the following (non-normative) table. In this table, "current" means the [browsing context](#) that the link or script is in, "parent" means the [parent browsing context](#) of the one the link or script is in, "top" means the [top-level browsing context](#) of the one the link or script is in, "new" means a new [top-level browsing context](#) or [auxiliary browsing context](#) is to be created, subject to various user preferences and user agent policies, "none" means that nothing will happen, and "maybe new" means the same as "new" if the "allow-popups" keyword is also specified on the [sandbox](#) attribute (or if the user overrode the sandboxing), and the same as "none" otherwise.

Keyword	Ordinary effect	Effect in an iframe with...	
		<code>sandbox=""</code>	<code>sandbox="allow-top-navigation"</code>
none specified, for links and form submissions	current	current	current
empty string	current	current	current
<code>_blank</code>	new	maybe new	maybe new
<code>_self</code>	current	current	current
<code>_parent</code> if there isn't a parent	current	current	current
<code>_parent</code> if parent is also top	parent/top	none	parent/top
<code>_parent</code> if there is one and it's not top	parent	none	none
<code>_top</code> if top is current	current	current	current
<code>_top</code> if top is not current	top	none	top
name that doesn't exist	new	maybe new	maybe new
name that exists and is a descendant	specified descendant	specified descendant	specified descendant
name that exists and is current	current	current	current

Keyword	Ordinary effect	Effect in an iframe with...	
		<code>sandbox=""</code>	<code>sandbox="allow-top-navigation"</code>
name that exists and is an ancestor that is top	specified ancestor	none	specified ancestor/top
name that exists and is an ancestor that is not top	specified ancestor	none	none
other name that exists with common top	specified	none	none
name that exists with different top, if familiar and one permitted sandboxed navigator	specified	specified	specified
name that exists with different top, if familiar but not one permitted sandboxed navigator	specified	none	none
name that exists with different top, not familiar	new	maybe new	maybe new

Most of the restrictions on sandboxed browsing contexts are applied by other algorithms, e.g., the [navigation](#) algorithm, not [the rules for choosing a browsing context given a browsing context name](#) given below.



An algorithm is **allowed to show a popup** if any of the following conditions is true:

- The [task](#) in which the algorithm is running is currently processing an [activation behavior](#) whose click event was [trusted](#).
- The [task](#) in which the algorithm is running is currently running the event listener for a [trusted](#) event whose type is in the following list:
 - change
 - click
 - dblclick
 - mouseup
 - reset
 - submit
- The [task](#) in which the algorithm is running was [queued](#) by an algorithm that was [allowed to show a popup](#), and the chain of such algorithms started within a user-agent defined timeframe.

EXAMPLE 602

For example, if a user clicked a button, it might be acceptable for a popup to result from that after 4 seconds, but it would likely not be acceptable for a popup to result from that after 4 hours.



The rules for choosing a browsing context given a browsing context name are as follows. The rules assume that they are being applied in the context of a [browsing context](#), as part of the execution of a [task](#).

1. If the given browsing context name is the empty string or `_self`, then the chosen browsing context must be the current one.
2. If the given browsing context name is `_parent`, then the chosen browsing context must be the [parent browsing context](#) of the current one, unless there isn't one, in which case the chosen browsing context must be the current browsing context.
3. If the given browsing context name is `_top`, then the chosen browsing context must be the [top-level browsing context](#) of the current one, if there is one, or else the current browsing context.
4. If the given browsing context name is not `_blank` and there exists a browsing context whose [name](#) is the same as the given browsing context name, and the current browsing context is familiar with that browsing context, and the user agent determines that the two browsing contexts are related enough that it is ok if they reach each other, then that

browsing context must be the chosen one. If there are multiple matching browsing contexts, the user agent should select one in some arbitrary consistent manner, such as the most recently opened, most recently focused, or more closely related.

5. Otherwise, a new browsing context is being requested, and what happens depends on the user agent's configuration and abilities — it is determined by the rules given for the first applicable option from the following list:

- ↪ **If the algorithm is not [allowed to show a popup](#) and the user agent has been configured to not show popups (i.e., the user agent has a "popup blocker" enabled)**

There is no chosen browsing context. The user agent may inform the user that a popup has been blocked.

- ↪ **If the current browsing context's [active document's active sandboxing flag set](#) has the [sandboxed auxiliary navigation browsing context flag set](#).**

Typically, there is no chosen browsing context.

The user agent may offer to create a new [top-level browsing context](#) or reuse an existing [top-level browsing context](#). If the user picks one of those options, then the designated browsing context must be the chosen one (the browsing context's name isn't set to the given browsing context name). The default behavior (if the user agent doesn't offer the option to the user, or if the user declines to allow a browsing context to be used) must be that there must not be a chosen browsing context.

⚠Warning! If this case occurs, it means that an author has explicitly sandboxed the document that is trying to open a link.

- ↪ **If the user agent has been configured such that in this instance it will create a new browsing context, and the browsing context is being requested as part of [following a hyperlink](#) whose [link types](#) include the [noreferrer](#) keyword**

A new [top-level browsing context](#) must be created. If the given browsing context name is not `_blank`, then the new top-level browsing context's name must be the given browsing context name (otherwise, it has no name). The chosen browsing context must be this new browsing context. The creation of such a [browsing context](#) is a new start for session storage.

NOTE:

If it is immediately [navigated](#), then the navigation will be done with [replacement enabled](#).

- ↪ **If the user agent has been configured such that in this instance it will create a new browsing context, and the [noreferrer](#) keyword doesn't apply**

A new [auxiliary browsing context](#) must be created, with the [opener browsing context](#) being the current one. If the given browsing context name is not `_blank`, then the new auxiliary browsing context's name must be the given browsing context name (otherwise, it has no name). The chosen browsing context must be this new browsing context.

NOTE:

If it is immediately [navigated](#), then the navigation will be done with [replacement enabled](#).

- ↪ **If the user agent has been configured such that in this instance it will reuse the current browsing context**

The chosen browsing context is the current browsing context.

- ↪ **If the user agent has been configured such that in this instance it will not find a browsing context**

There must not be a chosen browsing context.

User agent implementors are encouraged to provide a way for users to configure the user agent to always reuse the current browsing context.

If the current browsing context's [active document's active sandboxing flag set](#) has both the [sandboxed navigation browsing context flag](#) and [sandbox propagates to auxiliary browsing contexts flag set](#), and the chosen browsing context picked above, if any, is a new browsing context, then all the flags that are set in the current browsing context's [active document's active sandboxing flag set](#) when the new browsing context is created must be set in the new browsing context's [popup sandboxing flag set](#), and the current browsing context must be set as the new browsing context's [one per-](#)

[mitted sandboxed navigator](#).

§ 6.1.6. Script settings for browsing contexts

When the user agent is required to **set up a browsing context environment settings object**, given a JavaScript execution context *execution context*, it must run the following steps:

1. Let *realm* be the value of *execution context*'s Realm component.
2. Let *window* be *realm*'s [global object](#).
3. Let *url* be a copy of the [URL](#) of the [Document](#) with which *window* is associated.
4. Let *settings object* be a new [environment settings object](#) whose algorithms are defined as follows:

The [realm execution context](#)

Return *execution context*.

The [responsible browsing context](#)

Return the [browsing context](#) with which *window* is associated.

The [responsible event loop](#)

Return the [event loop](#) that is associated with the [unit of related similar-origin browsing contexts](#) to which *window*'s [browsing context](#) belongs.

The [responsible document](#)

Return the [Document](#) with which *window* is currently associated.

The [API URL character encoding](#)

Return the current [character encoding](#) of the [Document](#) with which *window* is currently associated.

The [API base URL](#)

Return the current [base URL](#) of the [Document](#) with which *window* is currently associated.

The [origin](#)

Return the [origin](#) of the [Document](#) with which *window* is currently associated.

The [creation URL](#)

Return *url*.

The [HTTPS state](#)

Return the [HTTPS state](#) of the [Document](#) with which *window* is currently associated.

5. Set *realm*'s `[[HostDefined]]` field to *settings object*.
6. Return *settings object*.

§ 6.2. Security infrastructure for [Window](#), [WindowProxy](#), and [Location](#) objects

Although typically objects cannot be accessed across origins, the web platform would not be true to itself if it did not have some legacy exceptions to that rule that the web depends upon.

§ 6.2.1. Integration with IDL

When [perform a security check](#) is invoked, with a *platformObject*, *realm*, *identifier*, and *type*, run these steps:

1. If *platformObject* is a [Window](#) or [Location](#) object, then:
 1. Repeat for each *e* that is an element of [CrossOriginProperties](#)(*platformObject*):
 1. If [SameValue](#)(*e*.[[Property]], *identifier*) is true, then:
 1. If *type* is "method" and *e* has neither `[[NeedsGet]]` nor `[[NeedsGet]]`, then return.

2. Otherwise, if *type* is "getter" and *e*.[[NeedsGet]] is true, then return.
 3. Otherwise, if *type* is "setter" and *e*.[[NeedsSet]] is true, then return.
2. If [IsPlatformObjectSameOrigin](#)(*platformObject*) is false, then throw a "[SecurityError](#)" [DOMException](#).

§ 6.2.2. Shared internal slot: [[CrossOriginPropertyDescriptorMap]]

[Window](#) and [Location](#) objects both have a [[CrossOriginPropertyDescriptorMap]] internal slot, whose value is initially an empty map.

NOTE:

The [[CrossOriginPropertyDescriptorMap]] internal slot contains a map with entries whose keys are (currentOrigin, objectOrigin, propertyKey)-tuples and values are property descriptors, as a memoization of what is visible to scripts when currentOrigin inspects a [Window](#) or [Location](#) object from objectOrigin. It is filled lazily by [CrossOriginGetOwnPropertyHelper](#), which consults it on future lookups.

User agents should allow a value held in the map to be garbage collected along with its corresponding key when nothing holds a reference to any part of the value. That is, as long as garbage collection is not observable.

EXAMPLE 603

For example, with

```
const href = Object.getOwnPropertyDescriptor(crossOriginLocation, "href").set
```

the value and its corresponding key in the map cannot be garbage collected as that would be observable.

User agents may have an optimization whereby they remove key-value pairs from the map when [document.domain](#) is set. This is not observable as [document.domain](#) cannot revisit an earlier value.

EXAMPLE 604

For example, setting [document.domain](#) to "example.com" on [www.example.com](#) means user agents can remove all key-value pairs from the map where part of the key is [www.example.com](#), as that can never be part of the origin again and therefore the corresponding value could never be retrieved from the map.

§ 6.2.3. Shared abstract operations

§ 6.2.3.1. CrossOriginProperties (*O*)

1. Assert: *O* is a [Location](#) or [Window](#) object.
2. If *O* is a [Location](#) object, then return


```
« {
  [[Property]]: "href",
  [[NeedsGet]]: false,
  [[NeedsSet]]: true
},
{
  [[Property]]: "replace"
} »
```
3. Let *crossOriginWindowProperties* be

```

« {
  [[Property]]: "window",
  [[NeedsGet]]: true,
  [[NeedsSet]]: false
},
{
  [[Property]]: "self",
  [[NeedsGet]]: true,
  [[NeedsSet]]: false
},
{
  [[Property]]: "location",
  [[NeedsGet]]: true,
  [[NeedsSet]]: true
},
{
  [[Property]]: "close"
},
{
  [[Property]]: "closed",
  [[NeedsGet]]: true,
  [[NeedsSet]]: false
},
{
  [[Property]]: "focus"
},
{
  [[Property]]: "blur"
},
{
  [[Property]]: "frames",
  [[NeedsGet]]: true,
  [[NeedsSet]]: false
},
{
  [[Property]]: "length",
  [[NeedsGet]]: true,
  [[NeedsSet]]: false
},
{
  [[Property]]: "top",
  [[NeedsGet]]: true,
  [[NeedsSet]]: false
},
{
  [[Property]]: "opener",
  [[NeedsGet]]: true,
  [[NeedsSet]]: false
},
{
  [[Property]]: "parent",
  [[NeedsGet]]: true,
  [[NeedsSet]]: false
},
{
  [[Property]]: "postMessage"
} »

```

2. Repeat for each *e* that is an element of the [child browsing context name property set](#):

1. Add { [[Property]]: *e* } as the last element of *crossOriginWindowProperties*.

3. Return *crossOriginWindowProperties*.

NOTE:

Indexed properties do not need to be safelisted as they are handled directly by the [WindowProxy](#) object.

§ 6.2.3.2. *IsPlatformObjectSameOrigin* (*O*)

1. Return true if the [current settings object](#)'s [origin](#) is [same origin-domain](#) with *O*'s [relevant settings object](#)'s [origin](#), and false otherwise.

§ 6.2.3.3. *CrossOriginGetOwnPropertyHelper* (*O*, *P*)**NOTE:**

If this abstract operation returns undefined and there is no custom behavior, the caller needs to throw a ["SecurityError" DOMException](#).

1. If *P* is [@@toStringTag](#), [@@hasInstance](#), or [@@isConcatSpreadable](#), then return [PropertyDescriptor](#) { [\[\[Value\]\]](#): undefined, [\[\[Writable\]\]](#): false, [\[\[Enumerable\]\]](#): false, [\[\[Configurable\]\]](#): true }.
2. Let *crossOriginKey* be a tuple consisting of the [current settings object](#)'s [origin](#)'s [effective domain](#), *O*'s relevant settings object's [origin](#)'s [effective domain](#), and *P*.
3. Repeat for each *e* that is an element of [CrossOriginProperties](#)(*O*):
 1. If [SameValue](#)(*e*.[\[\[Property\]\]](#), *P*) is true, then:
 1. If the value of the [\[\[CrossOriginPropertyDescriptorMap\]\]](#) internal slot of *O* contains an entry whose key is *crossOriginKey*, then return that entry's value.
 2. Let *originalDesc* be [OrdinaryGetOwnProperty](#)(*O*, *P*).
 3. Let *crossOriginDesc* be [CrossOriginPropertyDescriptor](#)(*e*, *originalDesc*).
 4. Create an entry in the value of the [\[\[CrossOriginPropertyDescriptorMap\]\]](#) internal slot of *O* with key *crossOriginKey* and value *crossOriginDesc*.
 5. Return *crossOriginDesc*.
4. Return undefined.

§ 6.2.3.3.1. *CrossOriginPropertyDescriptor* (*CrossOriginProperty*, *OriginalDesc*)

1. If *crossOriginProperty*.[\[\[NeedsGet\]\]](#) and *crossOriginProperty*.[\[\[NeedsSet\]\]](#) are absent, then:
 1. Let *value* be *originalDesc*.[\[\[Value\]\]](#).
 2. If [IsCallable](#)(*value*) is true, set *value* to [CrossOriginFunctionWrapper](#)(true, *value*).
 3. Return [PropertyDescriptor](#) { [\[\[Value\]\]](#): *value*, [\[\[Enumerable\]\]](#): false, [\[\[Writable\]\]](#): false, [\[\[Configurable\]\]](#): true }.
2. Otherwise:
 1. Let *crossOriginGet* be [CrossOriginFunctionWrapper](#)(*crossOriginProperty*.[\[\[NeedsGet\]\]](#), *originalDesc*.[\[\[Get\]\]](#)).
 2. Let *crossOriginSet* be [CrossOriginFunctionWrapper](#)(*crossOriginProperty*.[\[\[NeedsSet\]\]](#), *originalDesc*.[\[\[Set\]\]](#)).
 3. Return [PropertyDescriptor](#) { [\[\[Get\]\]](#): *crossOriginGet*, [\[\[Set\]\]](#): *crossOriginSet*, [\[\[Enumerable\]\]](#): false, [\[\[Configurable\]\]](#): true }.

§ 6.2.3.3.2. *CrossOriginFunctionWrapper* (*NeedsWrapping*, *FunctionToWrap*)

1. If *needsWrapping* is false, then return undefined.

2. Return a new [cross-origin wrapper function](#) whose `[[Wrapped]]` internal slot is *functionToWrap*.

A **cross-origin wrapper function** is an anonymous built-in function that has a `[[Wrapped]]` internal slot.

When a [cross-origin wrapper function](#) *F* is called with a list of arguments *argumentsList*, the following steps are taken:

1. Assert: *F* has a `[[Wrapped]]` internal slot that is a function.
2. Let *wrappedFunction* be the `[[Wrapped]]` internal slot of *F*.
3. Return [Call](#)(*wrappedFunction*, this, *argumentsList*).

NOTE:

Due to this being invoked from a different [origin](#), a [cross-origin wrapper function](#) will have a different value for `Function.prototype` from the function being wrapped. This follows from how JavaScript creates anonymous built-in functions.

§ 6.2.3.4. *CrossOriginGet* (*O*, *P*, *Receiver*)

1. Let *desc* be *O*.`[[GetOwnProperty]]`(*P*).
2. Assert: *desc* is not undefined.
3. If [IsDataDescriptor](#)(*desc*) is true, then return *desc*.`[[Value]]`.
4. Assert: [IsAccessorDescriptor](#)(*desc*) is true.
5. Let *getter* be *desc*.`[[Get]]`.
6. If *getter* is undefined, throw a "[SecurityError](#)" [DOMException](#).
7. Return [Call](#)(*getter*, *Receiver*).

§ 6.2.3.5. *CrossOriginSet* (*O*, *P*, *V*, *Receiver*)

1. Let *desc* be *O*.`[[GetOwnProperty]]`(*P*).
2. Assert: *desc* is not undefined.
3. If [IsAccessorDescriptor](#)(*desc*) is true, then:
 1. Let *setter* be *desc*.`[[Set]]`.
 2. If *setter* is undefined, return false.
 3. Perform [Call](#)(*setter*, *Receiver*, «V»).
 4. Return true.
4. Return false.

§ 6.2.3.6. *CrossOriginOwnPropertyKeys* (*O*)

1. Let *keys* be a new empty [List](#).
2. Repeat for each *e* that is an element of [CrossOriginProperties](#)(*O*):
 1. Add *e*.`[[Property]]` as the last element of *keys*.
3. Return *keys*.

§ 6.3. The window object

```

[PrimaryGlobal, LegacyUnenumerableNamedProperties]
/*sealed*/ interface Window : EventTarget {
// the current browsing context
[Unforgeable] readonly attribute WindowProxy window;
[Replaceable] readonly attribute WindowProxy self;
[Unforgeable] readonly attribute Document document;
attribute DOMString name;
[PutForwards=href, Unforgeable] readonly attribute Location location;
readonly attribute History history;
[Replaceable] readonly attribute BarProp locationbar;
[Replaceable] readonly attribute BarProp menubar;
[Replaceable] readonly attribute BarProp personalbar;
[Replaceable] readonly attribute BarProp scrollbars;
[Replaceable] readonly attribute BarProp statusbar;
[Replaceable] readonly attribute BarProp toolbar;
attribute DOMString status;
void close();
readonly attribute boolean closed;
void stop();
void focus();
void blur();

// other browsing contexts
[Replaceable] readonly attribute WindowProxy frames;
[Replaceable] readonly attribute unsigned long length;
[Unforgeable] readonly attribute WindowProxy top;
attribute any opener;
[Replaceable] readonly attribute WindowProxy parent;
readonly attribute Element? frameElement;
WindowProxy open(optional DOMString url = "about:blank", optional DOMString target = "_blank",
[TreatNullAs=EmptyString] optional DOMString features = "", optional boolean replace = false);
getter WindowProxy (unsigned long index);
getter object (DOMString name);

// the user agent
readonly attribute Navigator navigator;

// user prompts
void alert();
void alert(DOMString message);
boolean confirm(optional DOMString message = "");
DOMString? prompt(optional DOMString message = "", optional DOMString default = "");
void print();
any showModalDialog(DOMString url, optional any argument); // deprecated

unsigned long requestAnimationFrame(FrameRequestCallback callback);
void cancelAnimationFrame(unsigned long handle);
};
Window implements GlobalEventHandlers;
Window implements WindowEventHandlers;

callback FrameRequestCallback = void (DOMHighResTimeStamp time);

```

This definition is non-normative. Implementation requirements are given below this definition.

window . window

window . frames

window . self

These attributes all return **window**.

window . document

Returns the [Document](#) associated with **window**.

document . defaultView

Returns the `Window` object of the [active document](#).

The `window`, `frames`, and `self` IDL attributes must all return the [Window](#) object's [browsing context](#)'s [WindowProxy](#) object.

The `document` IDL attribute must return [the Window object's newest Document object](#).

NOTE:

The [Document](#) object associated with a `Window` object can change in exactly one case: when the [navigate](#) algorithm initializes a new [Document](#) object for the first page loaded in a [browsing context](#). In that specific case, the [Window](#) object of the original [about:blank](#) page is reused and gets a new [Document](#) object.

The `defaultView` IDL attribute of the [Document](#) interface must return the [Document](#)'s [browsing context](#)'s [WindowProxy](#) object, if there is one, or null otherwise.



For historical reasons, [Window](#) objects must also have a writable, configurable, non-enumerable property named `HTMLDocument` whose value is the [Document](#) interface object.

§ 6.3.1. APIs for creating and navigating browsing contexts by name

This definition is non-normative. Implementation requirements are given below this definition.

`window . open( [ url [, target [, features [, replace ] ] ] ] )`

Opens a window to show *url* (defaults to `about:blank`), and returns it. The *target* argument gives the name of the new window. If a window exists with that name already, it is reused. The *replace* attribute, if true, means that whatever page is currently open in that window will be removed from the window's session history. The *features* argument can be used to influence the rendering of the new window.

`window . name [ = value ]`

Returns the name of the window.

Can be set, to change the name.

`window . close()`

Closes the window.

`window . closed`

Returns true if the window has been closed, false otherwise.

`window . stop()`

Cancels the document load.

The `open()` method on `Window` objects provides a mechanism for [navigating](#) an existing [browsing context](#) or opening and navigating an [auxiliary browsing context](#).

When the method is invoked, the user agent must run the following steps:

1. Let *entry settings* be the [entry settings object](#) when the method was invoked.
2. Let *url* be the first argument.
3. Let *target* be the second argument.
4. Let *features* be the third argument.
5. Let *replace* be the fourth argument.
6. Let *source browsing context* be the [responsible browsing context](#) specified by *entry settings*.
7. If *target* is the empty string, let it be the string `"_blank"` instead.

8. If the user has indicated a preference for which [browsing context](#) to navigate, follow these substeps:

1. Let *target browsing context* be the [browsing context](#) indicated by the user.
2. If *target browsing context* is a new [top-level browsing context](#), let the *source browsing context* be set as *target browsing context*'s [one permitted sandboxed navigator](#).

EXAMPLE 605

For example, suppose there is a user agent that supports control-clicking a link to open it in a new tab. If a user clicks in that user agent on an element whose `onclick` handler uses the `window.open()` API to open a page in an iframe, but, while doing so, holds the control key down, the user agent could override the selection of the target browsing context to instead target a new tab.

Otherwise, apply [the rules for choosing a browsing context given a browsing context name](#) using *target* as the name and *source browsing context* as the context in which the algorithm is executed. If this results in there not being a chosen browsing context, then throw an `InvalidAccessError` exception and abort these steps. Otherwise, let *target browsing context* be the [browsing context](#) so obtained.

9. If *target browsing context* was just created, either as part of [the rules for choosing a browsing context given a browsing context name](#) or due to the user indicating a preference for navigating a new [top-level browsing context](#), then let *new* be true. Otherwise, let it be false.

10. Interpret *features* as defined in the CSSOM View specification. [\[CSSOM-VIEW\]](#)

11. If *url* is the empty string, run the appropriate steps from the following list:

If *new* is false

Jump to the step labeled *end*.

If *new* is true

Let *resource* be the [URL](#) "about:blank".

Otherwise, [parse](#) *url* relative to *entry settings*, and let *resource* be the [resulting URL record](#), if any. If the [parse a URL](#) algorithm failed, then run one of the following two steps instead:

- Let *resource* be a resource representing an inline error page.
- If *new* is false, jump to the step labeled *end*, otherwise, let *resource* be the [URL](#) "about:blank".

12. If *resource* is "about:blank" and *new* is true, [queue a task](#) to [fire a simple event](#) named `load` at *target browsing context*'s Window object, with *target override* set to *target browsing context*'s Window object's [Document](#) object.

Otherwise, [navigate](#) *target browsing context* to *resource*, with the [exceptions enabled flag](#) set. If *new* is true, then [re-
placement must be enabled](#) also. The [source browsing context](#) is *source browsing context*.

13. *End*: Return the `WindowProxy` object of *target browsing context*.



The `name` attribute of the Window object must, on getting, return the current [name](#) of the [browsing context](#), if one is set, or the empty string otherwise; and, on setting, set the [name](#) of the [browsing context](#) to the new value.

NOTE:

The name [gets reset](#) when the browsing context is navigated to another domain.



The `close()` method on Window objects should, if all the following conditions are met, [close](#) the [browsing context](#) *A*:

- The corresponding [browsing context](#) *A* is [script-closable](#).
- The [responsible browsing context](#) specified by the [incumbent settings object](#) is familiar with the [browsing context](#) *A*.
- The [responsible browsing context](#) specified by the [incumbent settings object](#) is [allowed to navigate](#) the [browsing con-](#)

[text](#) *A*.

A [browsing context](#) is **script-closable** if it is an [auxiliary browsing context](#) that was created by a script (as opposed to by an action of the user), or if it is a [top-level browsing context](#) whose [session history](#) contains only one [Document](#).

The **closed** attribute on Window objects must return true if the Window object's [browsing context](#) has been [discarded](#), and false otherwise.

The **stop()** method on Window objects should, if there is an existing attempt to [navigate](#) the [browsing context](#) and that attempt is not currently running the [unload a document](#) algorithm, cancel that [navigation](#); then, it must [abort](#) the [active document](#) of the [browsing context](#) of the Window object on which it was invoked.

§ 6.3.2. Accessing other browsing contexts

This definition is non-normative. Implementation requirements are given below this definition.

`window.length`

Returns the [number of child browsing contexts](#).

`window[index]`

Returns the indicated [child browsing context](#).

The **number of child browsing contexts** of a [Window](#) object *W* is the number of [child browsing contexts](#) that are [nested through](#) elements that are in a [Document](#) that is the [active document](#) of the [Window](#) object's [associated Document](#) object's [browsing context](#).

The **length** IDL attribute's getter must return the [number of child browsing contexts](#) of this [Window](#) object.

NOTE:

Indexed access to [child browsing contexts](#) is defined through the `[[GetOwnProperty]]` internal method of the [WindowProxy](#) object.

§ 6.3.3. Named access on the Window object

This definition is non-normative. Implementation requirements are given below this definition.

`window[name]`

Returns the indicated element or collection of elements.

As a general rule, relying on this will lead to brittle code. Which IDs end up mapping to this API can vary over time, as new features are added to the Web platform, for example. Instead of this, use [document.getElementById\(\)](#) or [document.querySelector\(\)](#).

The **child browsing context name property set** consists of the [browsing context names](#) of any [child browsing context](#) of the [active document](#) whose name is not the empty string, with duplicates omitted.

The Window interface [supports named properties](#). The [supported property names](#) at any moment consist of the following, in [tree order](#), ignoring later duplicates:

- the [child browsing context name property set](#).
- the value of the name content attribute for all `<a>`, `<applet>`, `<area>`, `<embed>`, `<form>`, `<frameset>`, `<img>`, and `<object>` elements in the [active document](#) that have a non-empty name content attribute, and
- the value of the id content attribute of any [HTML element](#) in the [active document](#) with a non-empty id content attribute.

To [determine the value of a named property](#) *name* when the **Window object is indexed for property retrieval**, the user agent must return the value obtained using the following steps:

1. Let *objects* be the list of [named objects](#) with the name *name* in the [active document](#).

NOTE:

There will be at least one such object, by definition.

2. If *objects* contains a [nested browsing context](#), then return the WindowProxy object of the [nested browsing context](#) corresponding to the first [browsing context container](#) in [tree order](#) whose [browsing context](#) is in *objects*, and abort these steps.
3. Otherwise, if *objects* has only one element, return that element and abort these steps.
4. Otherwise return an HTMLCollection rooted at the [Document](#) node, whose filter matches only [named objects](#) with the name *name*. (By definition, these will all be elements.)

Named objects with the name *name*, for the purposes of the above algorithm, are those that are either:

- [child browsing contexts](#) of the [active document](#) whose name is *name*,
- [<a>](#), [<applet>](#), [<area>](#), [<embed>](#), [<form>](#), [<frameset>](#), [](#), or [<object>](#) elements that have a name content attribute whose value is *name*, or
- [html elements](#) that have an id content attribute whose value is *name*.

§ 6.3.4. Garbage collection and browsing contexts

A [browsing context](#) has a strong reference to each of its [Documents](#) and its WindowProxy object, and the user agent itself has a strong reference to its [top-level browsing contexts](#).

A [Document](#) has a strong reference to its Window object.

NOTE:

A Window object has a strong reference to its [Document](#) object through its document attribute. Thus, references from other scripts to either of those objects will keep both alive. Similarly, both [Document](#) and Window objects have [implied strong references](#) to the WindowProxy object.

Each [script](#) has a strong reference to its [settings object](#), and each [environment settings object](#) has strong references to its [global object](#), [responsible browsing context](#), and [responsible document](#) (if any).

When a [browsing context](#) is to **discard a Document**, the user agent must run the following steps:

1. Set the [Document](#)'s *salvageable* state to false.
2. Run any [unloading document cleanup steps](#) for the [Document](#) that are defined by this specification and [other applicable specifications](#).
3. [Abort the Document](#).
4. Remove any [tasks](#) associated with the [Document](#) in any [task source](#), without running those tasks.
5. [Discard](#) all the [child browsing contexts](#) of the [Document](#).
6. Lose the strong reference from the [Document](#)'s [browsing context](#) to the [Document](#).

NOTE:

Whenever a [Document](#) object is [discarded](#), it is also removed from the list of the worker's Documents of each worker whose list contains that [Document](#).

When a **browsing context is discarded**, the strong reference from the user agent itself to the [browsing context](#) must be severed, and all the [Document](#) objects for all the entries in the [browsing context](#)'s session history must be [discarded](#) as well.

User agents may [discard top-level browsing contexts](#) at any time (typically, in response to user requests, e.g., when a user force-closes a window containing one or more [top-level browsing contexts](#)). Other [browsing contexts](#) must be discarded once their WindowProxy object is eligible for garbage collection.

§ 6.3.5. Closing browsing contexts

When the user agent is required to **close a browsing context**, it must run the following steps:

1. Let *specified browsing context* be the [browsing context](#) being closed.
2. [Prompt to unload](#) the [active document](#) of the *specified browsing context*. If the user [refused to allow the document to be unloaded](#), then abort these steps.
3. [Unload](#) the [active document](#) of the *specified browsing context* with the *recycle* parameter set to false.
4. Remove the *specified browsing context* from the user interface (e.g., close or hide its tab in a tabbed browser).
5. [Discard](#) the *specified browsing context*.

User agents should offer users the ability to arbitrarily [close](#) any [top-level browsing context](#).

§ 6.3.6. Browser interface elements

To allow Web pages to integrate with Web browsers, certain Web browser interface elements are exposed in a limited way to scripts in Web pages.

Each interface element is represented by a BarProp object:

```
interface BarProp {
  readonly attribute boolean visible;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

`window.locationbar.visible`

Returns true if the location bar is visible; otherwise, returns false.

`window menubar.visible`

Returns true if the menu bar is visible; otherwise, returns false.

`window.personalbar.visible`

Returns true if the personal bar is visible; otherwise, returns false.

`window.scrollbars.visible`

Returns true if the scroll bars are visible; otherwise, returns false.

`window.statusbar.visible`

Returns true if the status bar is visible; otherwise, returns false.

`window.toolbar.visible`

Returns true if the toolbar is visible; otherwise, returns false.

The **visible** attribute, on getting, must return either true or a value determined by the user agent to most accurately represent the visibility state of the user interface element that the object represents, as described below.

The following BarProp objects exist for each [Document](#) object in a [browsing context](#). Some of the user interface elements represented by these objects might have no equivalent in some user agents; for those user agents, except when otherwise specified, the object must act as if it was present and visible (i.e., its **visible** attribute must return true).

The location bar BarProp object

Represents the user interface element that contains a control that displays the [URL](#) of the [active document](#), or some similar interface concept.

The menu bar BarProp object

Represents the user interface element that contains a list of commands in menu form, or some similar interface concept.

The personal bar BarProp object

Represents the user interface element that contains links to the user's favorite pages, or some similar interface concept.

The scrollbar BarProp object

Represents the user interface element that contains a scrolling mechanism, or some similar interface concept.

The status bar BarProp object

Represents a user interface element found immediately below or after the document, as appropriate for the user's media, which typically provides information about ongoing network activity or information about elements that the user's pointing device is current indicating. If the user agent has no such user interface element, then the object may act as if the corresponding user interface element was absent (i.e., its `visible` attribute may return false).

The toolbar BarProp object

Represents the user interface element found immediately above or before the document, as appropriate for the user's media, which typically provides [session history](#) traversal controls (back and forward buttons, reload buttons, etc). If the user agent has no such user interface element, then the object may act as if the corresponding user interface element was absent (i.e., its `visible` attribute may return false).

The **locationbar** attribute must return [the location bar BarProp object](#).

The **menubar** attribute must return [the menu bar BarProp object](#).

The **personalbar** attribute must return [the personal bar BarProp object](#).

The **scrollbars** attribute must return [the scrollbar BarProp object](#).

The **statusbar** attribute must return [the status bar BarProp object](#).

The **toolbar** attribute must return [the toolbar BarProp object](#).



For historical reasons, the **status** attribute on the `Window` object must, on getting, return the last string it was set to, and on setting, must set itself to the new value. When the `Window` object is created, the attribute must be set to the empty string. It does not do anything else.

§ 6.3.7. The `WindowProxy` object

A **WindowProxy** is an exotic object that wraps a [Window](#) ordinary object, indirecting most operations through to the wrapped object. Each [browsing context](#) has an associated [WindowProxy](#) object. When the [browsing context](#) is [navigated](#), the [Window](#) object wrapped by the [browsing context](#)'s associated [WindowProxy](#) object is changed.

There is no [WindowProxy](#) interface object.

Every [WindowProxy](#) object has a `[[Window]]` internal slot representing the wrapped [Window](#) object.

NOTE:

Although [WindowProxy](#) is named as a "proxy", it does not do polymorphic dispatch on its target's internal methods as a real proxy would, due to a desire to reuse machinery between [WindowProxy](#) and [Location](#) objects. As long as the [Window](#) object remains an ordinary object this is unobservable and can be implemented either way.

EXAMPLE 606

In the following example, the variable `x` is set to the [WindowProxy](#) object returned by the `window` accessor on the [global object](#). All of the expressions following the assignment return true, because the [WindowProxy](#) object passes most operations through to the underlying ordinary [Window](#) object.

```
var x = window;
x instanceof Window; // true
x === this; // true
```

§ 6.3.7.1. The *WindowProxy* internal methods

The [WindowProxy](#) object internal methods are described in the subsections below.

§ 6.3.7.1.1. [\[\[GETPROTOTYPEOF\]\]](#) ()

1. Let *W* be the value of the [\[\[Window\]\]](#) internal slot of *this*.
2. If [IsPlatformObjectSameOrigin](#)(*W*) is true, then return ! [OrdinaryGetPrototypeOf](#)(*W*).
3. Return null.

§ 6.3.7.1.2. [\[\[SETPROTOTYPEOF\]\]](#) (*V*)

1. Return false.

§ 6.3.7.1.3. [\[\[ISEXTENSIBLE\]\]](#) ()

1. Return true.

§ 6.3.7.1.4. [\[\[PREVENTEXTENSIONS\]\]](#) ()

1. Return false.

§ 6.3.7.1.5. [\[\[GETOWNPROPERTY\]\]](#) (*P*)

1. Let *W* be the value of the [\[\[Window\]\]](#) internal slot of *this*.
2. If *P* is an [array index property name](#), then:
 1. Let *index* be [ToUint32](#)(*P*).
 2. Let *maxProperties* be the [number of child browsing contexts](#) of *W*.
 3. Let *value* be undefined.
 4. If *maxProperties* is greater than 0 and *index* is less than *maxProperties*, then:
 1. Set *value* to the [WindowProxy](#) object of the *index*th [child browsing context](#) of the [Document](#) that is [nested through](#) an element that is in *W*'s [Document](#), sorted in the order that the elements nesting those [browsing contexts](#) were most recently inserted into the [Document](#), the [WindowProxy](#) object of the most recently inserted [browsing context container](#)'s [nested browsing context](#) being last.
 5. Return [PropertyDescriptor](#){ [\[\[Value\]\]](#): *value*, [\[\[Writable\]\]](#): false, [\[\[Enumerable\]\]](#): false, [\[\[Configurable\]\]](#): true }.
3. If [IsPlatformObjectSameOrigin](#)(*W*) is true, then return [OrdinaryGetOwnProperty](#)(*W*, *P*).

NOTE:

This violates JavaScript's internal method invariants.

4. Let *property* be [CrossOriginGetOwnPropertyHelper](#)(*W*, *P*).
5. If *property* is not undefined, return *property*.
6. If *property* is undefined and *P* is in the [child browsing context name property set](#), then:
 1. Let *value* be the [WindowProxy](#) object of the [named object](#) with the name *P*.
 2. Return [PropertyDescriptor](#){ [\[\[Value\]\]](#): *value*, [\[\[Enumerable\]\]](#): false, [\[\[Writable\]\]](#): false, [\[\[Configurable\]\]](#): true }.
7. Throw a "[SecurityError](#)" [DOMException](#).

§ 6.3.7.1.6. `[[DEFINEOWNPROPERTY]]` (*P*, *DESC*)

1. If *P* is an [array index property name](#), return false.
2. Let *W* be the value of the `[[Window]]` internal slot of *this*.
3. If `IsPlatformObjectSameOrigin(W)` is true, then return `OrdinaryDefineOwnProperty(W, P, Desc)`.

NOTE:

See above about how this violates JavaScript's internal method invariants.

4. Return false.

§ 6.3.7.1.7. `[[GET]]` (*P*, *RECEIVER*)

1. Let *W* be the value of the `[[Window]]` internal slot of *this*.
2. If `IsPlatformObjectSameOrigin(W)` is true, then return `OrdinaryGet(this, P, Receiver)`.
3. Return ? `CrossOriginGet(this, P, Receiver)`.

§ 6.3.7.1.8. `[[SET]]` (*P*, *V*, *RECEIVER*)

1. Let *W* be the value of the `[[Window]]` internal slot of *this*.
2. If `IsPlatformObjectSameOrigin(W)` is true, then return `OrdinarySet(W, this, Receiver)`.
3. Return `CrossOriginSet(this, P, V, Receiver)`.

§ 6.3.7.1.9. `[[DELETE]]` (*P*)

1. If *P* is an [array index property name](#), return false.
2. Let *W* be the value of the `[[Window]]` internal slot of *this*.
3. If `IsPlatformObjectSameOrigin(W)` is true, then return `OrdinaryDelete(W, P)`.
4. Return false.

§ 6.3.7.1.10. `[[OWNPROPERTYKEYS]]` ()

1. Let *W* be the value of the `[[Window]]` internal slot of *this*.
2. Let *keys* be a new empty [List](#).
3. Let *maxProperties* be the [number of child browsing contexts](#) of *W*.
4. Let *index* be 0.
5. Repeat while *index* < *maxProperties*,
 1. Add ! `ToString(index)` as the last element of *keys*.
 2. Increment *index* by 1.
6. If `IsPlatformObjectSameOrigin(W)` is true, then return the concatenation of *keys* and ! `OrdinaryOwnPropertyKeys(W)`.
7. Return the concatenation of *keys* and ! `CrossOriginOwnPropertyKeys(W)`.

§ 6.4. Origin

Origins are the fundamental currency of the Web's security model. Two actors in the Web platform that share an origin are assumed to trust each other and to have the same authority. Actors with differing origins are considered potentially hostile versus each other, and are isolated from each other to varying degrees.

EXAMPLE 607

For example, if Example Bank's Web site, hosted at `bank.example.com`, tries to examine the DOM of Example Charity's Web site, hosted at `charity.example.org`, a "[SecurityError](#)" [DOMException](#) will be raised.



An **origin** is one of the following:

An opaque origin

An internal value, with no serialisation, for which the only meaningful operation is testing for equality.

A tuple origin

A [tuple](#) consists of:

- A **scheme** (a [scheme](#)).
- A **host** (a [host](#)).
- A **port** (a [port](#)).
- A **domain** (null or a [domain](#)). Null unless stated otherwise.

NOTE:

Origins can be shared, e.g., among multiple [Document](#) objects. Furthermore, [origins](#) are generally immutable. Only the [domain](#) of a [tuple origin](#) can be changed, and only through the [document.domain](#) API.

The **effective domain** of an [origin](#) *origin* is computed as follows:

1. If *origin* is an [opaque origin](#), then return *origin*.
2. If *origin*'s [domain](#) is non-null, then return *origin*'s [domain](#).
3. Return *origin*'s [host](#).

Various specification objects are defined to have an [origin](#). These [origins](#) are determined as follows:

For [Document](#) objects

↪ If the [Document](#)'s **active sandboxing flag set** has its **sandboxed origin browsing context flag set**
A unique [opaque origin](#) is assigned when the [Document](#) is created.

↪ If the [Document](#)'s **URL's scheme** is a **network scheme**
A copy of the [Document](#)'s [URL](#)'s [origin](#) assigned when the [Document](#) is created.

NOTE:

The [document.open\(\)](#) method can change the [Document](#)'s [URL](#) to "about:blank". Therefore the [origin](#) is assigned when the [Document](#) is created.

↪ If the [Document](#) is the initial "about:blank" document
The one it was assigned when its browsing context was created.

↪ If the [Document](#) is a non-initial "about:blank" document

↪ If the [Document](#) was generated from a **data: URL** found in another [Document](#) or in a script
The [origin](#) of the [incumbent settings object](#) when the [navigate](#) algorithm was invoked, or, if no [script](#) was involved, of the [node document](#) of the element that initiated the [navigation](#) to that [URL](#).

↪ If the [Document](#) was created as part of the processing for **javascript: URLs**
The [origin](#) of the [active document](#) of the [browsing context](#) being navigated when the [navigate](#) algorithm was invoked.

↪ If the [Document](#) is an **iframe srcdoc document**
The [origin](#) of the [Document](#)'s browsing context's [browsing context container](#)'s [node document](#).

- ↪ If the [Document](#) was obtained in some other manner (e.g., a [data: URL](#) typed in by the user or that was returned as the location of a redirect, a [Document](#) created using the [createDocument\(\)](#) API, etc)

The default behavior as defined in the DOM standard applies. [\[DOM\]](#).

NOTE:

The [origin](#) is a unique [opaque origin](#) assigned when the [Document](#) is created.

For images of [](#) elements

- ↪ If the image data is [CORS-cross-origin](#)
A unique [opaque origin](#) assigned when the image is created.
- ↪ If the image data is [CORS-same-origin](#)
The [](#) element's [node document](#)'s [origin](#).

For [<audio>](#) and [<video>](#) elements

- ↪ If the [media data](#) is [CORS-cross-origin](#)
A unique [opaque origin](#) assigned when the [media data](#) is fetched.
- ↪ If the [media data](#) is [CORS-same-origin](#)
The [media element](#)'s [node document](#)'s [origin](#).

For fonts

For a downloadable Web font it is a copy of the [origin](#) of the [URL record](#) used to obtain the font (after any redirects).
[\[CSS-FONTS-3\]](#) [\[CSS-FONT-LOADING-3\]](#)

For a locally installed system font it is the [origin](#) of the [Document](#) in which that font is being used.

Other specifications can override the above definitions by themselves specifying the origin of a particular [Document](#) object, image, [media element](#), or font.



The **Unicode serialization of an origin** is the string obtained by applying the following algorithm to the given [origin](#) *origin*:

1. If *origin* is an [opaque origin](#), then return "null".
2. Let *host* be *origin*'s [host](#).
3. Let *unicodeHost* be *host* if *host* is not a [domain](#), and the result of applying [domain to Unicode](#) to *host* otherwise.
4. Let *unicodeOrigin* be a new [tuple origin](#) consisting *origin*'s [scheme](#), *unicodeHost*, and *origin*'s [port](#).
5. Return the [ASCII serialization of an origin](#), given *unicodeOrigin*.

NOTE:

The name [ASCII serialization of an origin](#) is misleading, as it merely serialises an origin, which are all ASCII by default due to the [URL parser](#).

EXAMPLE 608

The [Unicode serialization](#) of ("https", "xn--maraa-rta.example", null, null) is "https://maraña.example".

The **ASCII serialization of an origin** is the string obtained by applying the following algorithm to the given [origin](#) *origin*:

1. If *origin* is an [opaque origin](#), then return "null".
2. Otherwise, let *result* be *origin*'s [scheme](#).
3. Append "://" to *result*.
4. Append *origin*'s [host, serialized](#), to *result*.
5. If *origin*'s [port](#) is non-null, append a U+003A COLON character (:), and *origin*'s [port, serialized](#), to *result*.

6. Return *result*.

Two [origins](#) *A* and *B* are said to be **same origin** if the following algorithm returns true:

1. If *A* and *B* are the same [opaque origin](#), then return true.
2. If *A* and *B* are both [tuple origins](#), and their [schemes](#), [hosts](#), and [ports](#) are identical, then return true.
3. Return false.

Two [origins](#) *A* and *B* are said to be **same origin-domain** if the following algorithm returns true:

1. If *A* and *B* are the same [opaque origin](#), then return true.
2. If *A* and *B* are both [tuple origins](#), run these substeps:
 1. If *A* and *B*'s [schemes](#) are identical, and their [domains](#) are identical and non-null, then return true.
 2. Otherwise, if *A* and *B* are [same origin](#) and their [domains](#) are identical and null, then return true.
3. Return false.

EXAMPLE 609

The following table shows how *A* and *B* are related:

A	B	same origin	same origin-domain
("https", "example.org", null, null)	("https", "example.org", null, null)	✓	✓
("https", "example.org", 314, "example.org")	("https", "example.org", 420, "example.org")	✗	✓
("https", "example.org", null, null)	("https", "example.org", null, "example.org")	✓	✗
("https", "example.org", null, "example.org")	("http", "example.org", null, "example.org")	✗	✗

§ 6.4.1. Relaxing the same-origin restriction

This definition is non-normative. Implementation requirements are given below this definition.

***document*.domain** [= *domain*]

Returns the current domain used for security checks.

Can be set to a value that removes subdomains, to change the [origin](#)'s [domain](#) to allow pages on other subdomains of the same domain (if they do the same thing) to access each other. (Can't be set in sandboxed [<i>iframe</i>s](#).)

The **domain** attribute's getter must run these steps:

1. If this [Document](#) object does not have a [browsing context](#), then return the empty string.
2. Let *effectiveDomain* be this [Document](#)'s [origin](#)'s [effective domain](#).
3. If *effectiveDomain* is an [opaque origin](#), then return the empty string.
4. Return *effectiveDomain*, [serialised](#).

The **domain** attribute on setting must run these steps:

1. If this [Document](#) object has no [browsing context](#), throw a "[SecurityError](#)" [DOMException](#).
2. If this [Document](#) object's [active sandboxing flag set](#) has its [sandboxed document.domain browsing context flag](#) set, then throw a "[SecurityError](#)" [DOMException](#).
3. If the given value is the empty string, then throw a "[SecurityError](#)" [DOMException](#).

4. Let *host* be the result of [parsing](#) the given value.
5. If *host* is failure, then throw a "[SecurityError](#)" [DOMException](#).
6. Let *effectiveDomain* be this [Document](#) object's [origin](#)'s [effective domain](#).
7. If *host* is not [equal](#) to *effectiveDomain*, then run these substeps:
 1. If *host* or *effectiveDomain* is not [domain](#), then throw a "[SecurityError](#)" [DOMException](#).

NOTE:

This is meant to exclude [hosts](#) that are an [IPv4 address](#) or an [IPv6 address](#).

2. If *host*, prefixed by a U+002E FULL STOP (.), does not exactly match the *effectiveDomain*, then throw a "[SecurityError](#)" [DOMException](#).
3. If *host* matches a suffix in the Public Suffix List, or, if *host*, prefixed by a U+002E FULL STOP (.), matches the end of a suffix in the Public Suffix List, then throw a "[SecurityError](#)" [DOMException](#). [\[PSL\]](#)

Suffixes must be compared after applying the [host parser](#) algorithm. [\[URL\]](#)

8. Set *origin*'s [domain](#) to *host*.

NOTE:

The [document.domain](#) attribute is used to enable pages on different hosts of a domain to access each others' DOMs.

⚠Warning! Do not use the [document.domain](#) attribute when using shared hosting. If an untrusted third party is able to host an HTTP server at the same IP address but on a different port, then the same-origin protection that normally protects two different sites on the same host will fail, as the ports are ignored when comparing origins after the [document.domain](#) attribute has been used.

§ 6.5. Sandboxing

A **sandboxing flag set** is a set of zero or more of the following flags, which are used to restrict the abilities that potentially untrusted resources have:

The *sandboxed navigation browsing context flag*

This flag [prevents content from navigating browsing contexts other than the sandboxed browsing context itself](#) (or browsing contexts further nested inside it), [auxiliary browsing contexts](#) (which are protected by the [sandboxed auxiliary navigation browsing context flag](#) defined next), and the [top-level browsing context](#) (which is protected by the [sandboxed top-level navigation browsing context flag](#) defined below).

If the [sandboxed auxiliary navigation browsing context flag](#) is not set, then in certain cases the restrictions nonetheless allow popups (new [top-level browsing contexts](#)) to be opened. These [browsing contexts](#) always have **one permitted sandboxed navigator**, set when the browsing context is created, which allows the [browsing context](#) that created them to actually navigate them. (Otherwise, the [sandboxed navigation browsing context flag](#) would prevent them from being navigated even if they were opened.)

The *sandboxed auxiliary navigation browsing context flag*

This flag [prevents content from creating new auxiliary browsing contexts](#), e.g., using the `target` attribute, the `window.open()` method, or the `showModalDialog()` method.

The *sandboxed top-level navigation browsing context flag*

This flag [prevents content from navigating their top-level browsing context](#) and [prevents content from closing their top-level browsing context](#).

When the [sandboxed top-level navigation browsing context flag](#) is *not* set, content can navigate its [top-level browsing context](#), but other [browsing contexts](#) are still protected by the [sandboxed navigation browsing context flag](#) and possibly the [sandboxed auxiliary navigation browsing context flag](#).

The *sandboxed plugins browsing context flag*

This flag prevents content from instantiating [plugins](#), whether using the `<embed>` element, the `<object>` element, the `<applet>` element, or through [navigation](#) of a [nested browsing context](#), unless those [plugins](#) can be [secured](#).

The *sandboxed origin browsing context flag*

This flag [forces content into a unique origin](#), thus preventing it from accessing other content from the same [origin](#).

This flag also [prevents script from reading from or writing to the `document.cookie` IDL attribute](#), and blocks access to `localStorage`. [\[WEBSTORAGE\]](#)

The *sandboxed forms browsing context flag*

This flag [blocks form submission](#).

The *sandboxed pointer lock browsing context flag*

This flag disables the Pointer Lock API. [\[POINTERLOCK\]](#)

The *sandboxed scripts browsing context flag*

This flag [blocks script execution](#).

The *sandboxed automatic features browsing context flag*

This flag blocks features that trigger automatically, such as [automatically playing a video](#) or [automatically focusing a form control](#).

The *sandboxed storage area URLs flag*

This flag prevents URL schemes that use storage areas from being able to access the origin's data.

The *sandboxed fullscreen browsing context flag*

This flag prevents content from using the `requestFullscreen()` method.

The *sandboxed `document.domain` browsing context flag*

This flag prevents content from using the [document.domain](#) setter.

The *sandbox propagates to auxiliary browsing contexts flag*

This flag prevents content from escaping the sandbox by ensuring that any [auxiliary browsing context](#) it creates inherits the content's [active sandboxing flag set](#).

The *sandboxed modals flag*

This flag prevents content from using any of the following features to produce modal dialogs:

- `window.alert()`
- `window.confirm()`
- `window.print()`
- `window.prompt()`
- `window.showModalDialog()`
- the `beforeunload` event

When the user agent is to **parse a sandboxing directive**, given a string *input*, a [sandboxing flag set](#) *output*, and optionally an *allow fullscreen flag*, it must run the following steps:

1. [Split *input* on spaces](#), to obtain *tokens*.
2. Let *output* be empty.
3. Add the following flags to *output*:
 - The [sandboxed navigation browsing context flag](#).
 - The [sandboxed auxiliary navigation browsing context flag](#), unless *tokens* contains the **allow-popups** keyword.

- The [sandboxed top-level navigation browsing context flag](#), unless `tokens` contains the **allow-top-navigation** keyword.
- The [sandboxed plugins browsing context flag](#).
- The [sandboxed origin browsing context flag](#), unless the `tokens` contains the **allow-same-origin** keyword.

The **allow-same-origin** keyword is intended for two cases.

First, it can be used to allow content from the same site to be sandboxed to disable scripting, while still allowing access to the DOM of the sandboxed content.

Second, it can be used to embed content from a third-party site, sandboxed to prevent that site from opening pop-up windows, etc, without preventing the embedded page from communicating back to its originating site, using the database APIs to store data, etc.

- The [sandboxed forms browsing context flag](#), unless `tokens` contains the **allow-forms** keyword.
- The [sandboxed pointer lock browsing context flag](#), unless `tokens` contains the **allow-pointer-lock** keyword.
- The [sandboxed scripts browsing context flag](#), unless `tokens` contains the **allow-scripts** keyword.
- The [sandboxed automatic features browsing context flag](#), unless `tokens` contains the **allow-scripts** keyword (defined above).

NOTE:

This flag is relaxed by the same keyword as scripts, because when scripts are enabled these features are trivially possible anyway, and it would be unfortunate to force authors to use script to do them when sandboxed rather than allowing them to use the declarative features.

- The [sandboxed storage area URLs flag](#).
- The [sandboxed fullscreen browsing context flag](#), unless the `allow fullscreen flag` was passed to the [parse a sandboxing directive](#) flag.
- The [sandboxed document.domain browsing context flag](#).
- The [sandbox propagates to auxiliary browsing contexts flag](#), unless `tokens` contains the **allow-popups-to-escape-sandbox** keyword.
- The [sandboxed modals flag](#), unless `tokens` contains the **allow-modals** keyword.



Every [top-level browsing context](#) has a **popup sandboxing flag set**, which is a [sandboxing flag set](#). When a [browsing context](#) is created, its [popup sandboxing flag set](#) must be empty. It is populated by [the rules for choosing a browsing context given a browsing context name](#).

Every [nested browsing context](#) has an **iframe sandboxing flag set**, which is a [sandboxing flag set](#). Which flags in a [nested browsing context](#)'s [iframe sandboxing flag set](#) are set at any particular time is determined by the `<iframe>` element's [sandbox](#) attribute.

Every [Document](#) has an **active sandboxing flag set**, which is a [sandboxing flag set](#). When the [Document](#) is created, its [active sandboxing flag set](#) must be empty. It is populated by the [navigation algorithm](#).

Every resource that is obtained by the [navigation algorithm](#) has a **forced sandboxing flag set**, which is a [sandboxing flag set](#). A resource by default has no flags set in its [forced sandboxing flag set](#), but other specifications can define that certain flags are set.

NOTE:

In particular, the [forced sandboxing flag set](#) is used by the Content Security Policy specification. [\[CSP3\]](#)



When a user agent is to **implement the sandboxing** for a [Document](#), it must populate [Document's active sandboxing flag set](#) with the union of the flags that are present in the following [sandboxing flag sets](#) at the time the [Document](#) object is created:

- If the [Document's browsing context](#) is a [top-level browsing context](#), then: the flags set on the [browsing context's popup sandboxing flag set](#).
- If the [Document's browsing context](#) is a [nested browsing context](#), then: the flags set on the [browsing context's iframe sandboxing flag set](#).
- If the [Document's browsing context](#) is a [nested browsing context](#), then: the flags set on the [browsing context's parent browsing context's active document's active sandboxing flag set](#).
- The flags set on the [Document's resource's forced sandboxing flag set](#), if it has one.

§ 6.6. Session history and navigation

§ 6.6.1. The session history of browsing contexts

The sequence of [Documents](#) in a [browsing context](#) is its **session history**. Each [browsing context](#), including [nested browsing contexts](#), has a distinct session history. A [browsing context's](#) session history consists of a flat list of [session history entries](#). Each **session history entry** consists, at a minimum, of a [URL](#), and each entry may in addition have a [state object](#), a title, a [Document](#) object, form data, a [scroll restoration mode](#), a scroll position, and other information associated with it.

NOTE:

Each entry, when first created, has a [Document](#). However, when a [Document](#) is not [active](#), it's possible for it to be [discarded](#) to free resources. The [URL](#) and other data in a [session history entry](#) is then used to bring a new [Document](#) into being to take the place of the original, should the user agent find itself having to reactivate that [Document](#).

NOTE:

Titles associated with [session history entries](#) need not have any relation with the current title of the [Document](#). The title of a [session history entry](#) is intended to explain the state of the document at that point, so that the user can navigate the document's history.

URLs without associated [state objects](#) are added to the session history as the user (or script) navigates from page to page.



Each [Document](#) object in a [browsing context's session history](#) is associated with a unique History object which must all model the same underlying [session history](#).

The **history** attribute of the Window interface must return the object implementing the History interface for that [Window](#) object's newest [Document](#).



A **state object** is an object representing a user interface state.

Pages can [add state objects](#) to the session history. These are then [returned to the script](#) when the user (or script) goes back in the history, thus enabling authors to use the "navigation" metaphor even in one-page applications.

[State objects](#) are intended to be used for two main purposes: first, storing a prepared description of the state in the [URL](#) so that in the simple case an author doesn't have to do the parsing (though one would still need the parsing for handling [URLs](#) passed around by users, so it's only a minor optimization), and second, so that the author can store state that one wouldn't store in the URL because it only applies to the current [Document](#) instance and it would have to be reconstructed if a new [Document](#) were opened.

An example of the latter would be something like keeping track of the precise coordinate from which a pop-up `div` was made to animate, so that if the user goes back, it can be made to animate to the same location. Or alternatively, it could be used to keep a pointer into a cache of data that would be fetched from the server based on the information in the [URL](#), so that when going back and forward, the information doesn't have to be fetched again.



At any point, one of the entries in the session history is the **current entry**. This is the entry representing the [active document](#) of the [browsing context](#). Which entry is the [current entry](#) is changed by the algorithms defined in this specification, e.g., during [session history traversal](#).

NOTE:

The [current entry](#) is usually an entry for the [address](#) of the [Document](#). However, it can also be one of the entries for [state objects](#) added to the history by that document.

An **entry with persisted user state** is one that also has user-agent defined state. This specification does not specify what kind of state can be stored.

EXAMPLE 610

For example, some user agents might want to persist the scroll position, or the values of form controls.

NOTE:

User agents that persist the value of form controls are encouraged to also persist their directionality (the value of the element's `dir` attribute). This prevents values from being displayed incorrectly after a history traversal when the user had originally entered the values with an explicit, non-default directionality.

An entry's **scroll restoration mode** indicates whether the user agent should restore the persisted scroll position (if any) when traversing to it. The scroll restoration mode may be one of the following:

"*auto*"

The user agent is responsible for restoring the scroll position upon navigation.

"*manual*"

The page is responsible for restoring the scroll position and the user agent does not attempt to do so automatically

If unspecified, the [scroll restoration mode](#) of a new entry must be set to "auto".

Entries that consist of [state objects](#) share the same [Document](#) as the entry for the page that was active when they were added.

Contiguous entries that differ just by fragment identifier also share the same [Document](#).

NOTE:

All entries that share the same [Document](#) (and that are therefore merely different states of one particular document) are contiguous by definition.

Each [Document](#) in a [browsing context](#) can also have a **latest entry**. This is the entry for that [Document](#) to which the [browsing context](#)'s [session history](#) was most recently traversed. When a [Document](#) is created, it initially has no [latest entry](#).

User agents may [discard](#) the [Document](#) objects of entries other than the [current entry](#) that are not referenced from any script, reloading the pages afresh when the user or script navigates back to such pages. This specification does not specify when user agents should discard [Document](#) objects and when they should cache them.

Entries that have had their [Document](#) objects discarded must, for the purposes of the algorithms given below, act as if they had not. When the user or script navigates back or forwards to a page which has no in-memory DOM objects, any other entries that shared the same [Document](#) object with it must share the new object as well.

§ 6.6.2. The History interface

```
enum ScrollRestoration { "auto", "manual" };

interface History {
  readonly attribute unsigned long length;
  attribute ScrollRestoration scrollRestoration;
  readonly attribute any state;
  void go(optional long delta = 0);
  void back();
  void forward();
  void pushState(any data, DOMString title, optional DOMString? url = null);
  void replaceState(any data, DOMString title, optional DOMString? url = null);
};
```

This definition is non-normative. Implementation requirements are given below this definition.

`window.history.length`

Returns the number of entries in the [joint session history](#).

`window.history.scrollRestoration` [= *value*]

Returns the [scroll restoration mode](#) of the current entry in the [session history](#).

Can be set, to change the [scroll restoration mode](#) of the current entry in the [session history](#).

`window.history.state`

Returns the current [state object](#).

`window.history.go` [*delta*]

Goes back or forward the specified number of steps in the [joint session history](#).

A zero delta will reload the current page.

If the delta is out of range, does nothing.

`window.history.back`()

Goes back one step in the [joint session history](#).

If there is no previous page, does nothing.

`window.history.forward`()

Goes forward one step in the [joint session history](#).

If there is no next page, does nothing.

`window.history.pushState`(*data*, *title* [, *url*])

Pushes the given data onto the session history, with the given title, and, if provided and not null, the given URL.

`window.history.replaceState`(*data*, *title* [, *url*])

Updates the current entry in the session history to have the given data, title, and, if provided and not null, URL.

The **joint session history** of a [top-level browsing context](#) is the union of all the [session histories](#) of all [browsing contexts](#) of all the [fully active Document](#) objects that share that [top-level browsing context](#), with all the entries that are [current entries](#) in their respective [session histories](#) removed except for the [current entry of the joint session history](#).

The **current entry of the joint session history** is the entry that most recently became a [current entry](#) in its [session history](#).

Entries in the [joint session history](#) are ordered chronologically by the time they were added to their respective [session histories](#). Each entry has an index; the earliest entry has index 0, and the subsequent entries are numbered with consecutively in-

creasing integers (1, 2, 3, etc).

NOTE:

Since each [Document](#) in a [browsing context](#) might have a different [event loop](#), the actual state of the [joint session history](#) can be somewhat nebulous. For example, two sibling `<iframe>` elements could both [traverse](#) from one unique origin to another at the same time, so their precise order might not be well-defined; similarly, since they might only find out about each other later, they might disagree about the length of the [joint session history](#).

The **length** attribute of the History interface, on getting, must return the number of entries in the [top-level browsing context's joint session history](#). If this History object is associated with a [Document](#) that is not [fully active](#), getting must instead throw a "[SecurityError](#)" [DOMException](#).

The actual entries are not accessible from script.

The **scrollRestoration** attribute of the History interface, on getting, must return the [scroll restoration mode](#) of the current entry in the [session history](#). On setting, the [scroll restoration mode](#) of the current entry in the [session history](#) must be set to the new value. If this History object is associated with a [Document](#) that is not [fully active](#), both getting and setting must instead throw a "[SecurityError](#)" [DOMException](#).

The **state** attribute of the History interface, on getting, must return the last value it was set to by the user agent. If this History object is associated with a [Document](#) that is not [fully active](#), getting must instead throw a [SecurityError](#) [DOMException](#). Initially, its value must be null.

When the **go(delta)** method is invoked, if *delta* is zero, the user agent must act as if the `location.reload()` method was called instead. Otherwise, the user agent must [traverse the history by a delta](#) whose value is *delta*. If this History object is associated with a [Document](#) that is not [fully active](#), invoking must instead throw a "[SecurityError](#)" [DOMException](#).

When the **back()** method is invoked, the user agent must [traverse the history by a delta](#) -1 . If this History object is associated with a [Document](#) that is not [fully active](#), invoking must instead throw a "[SecurityError](#)" [DOMException](#).

When the **forward()** method is invoked, the user agent must [traverse the history by a delta](#) $+1$. If this History object is associated with a [Document](#) that is not [fully active](#), invoking must instead throw a "[SecurityError](#)" [DOMException](#).



Each [top-level browsing context](#) has a **session history traversal queue**, initially empty, to which [tasks](#) can be added.

Each [top-level browsing context](#), when created, must begin running the following algorithm, known as the **session history event loop** for that [top-level browsing context](#), [in parallel](#):

1. Wait until this [top-level browsing context's session history traversal queue](#) is not empty.
2. Pull the first [task](#) from this [top-level browsing context's session history traversal queue](#), and execute it.
3. Return to the first step of this algorithm.

The [session history event loop](#) helps coordinate cross-browsing-context transitions of the [joint session history](#): since each [browsing context](#) might, at any particular time, have a different [event loop](#) (this can happen if the user agent has more than one [event loop](#) per [unit of related browsing contexts](#)), transitions would otherwise have to involve cross-event-loop synchronization.



To **traverse the history by a delta** *delta*, the user agent must append a [task](#) to this [top-level browsing context's session history traversal queue](#), the [task](#) consisting of running the following steps:

1. If the index of the [current entry of the joint session history](#) plus *delta* is less than zero or greater than or equal to the number of items in the [joint session history](#), then abort these steps.
2. Let *specified entry* be the entry in the [joint session history](#) whose index is the sum of *delta* and the index of the [current entry of the joint session history](#).

3. Let *specified browsing context* be the [browsing context](#) of the *specified entry*.
4. If the *specified browsing context*'s [active document](#)'s [unload a document](#) algorithm is currently running, abort these steps.
5. [Queue a task](#) that consists of running the following substeps. The relevant [event loop](#) is that of the *specified browsing context*'s [active document](#). The [task source](#) for the queued task is the [history traversal task source](#).
 1. If there is an ongoing attempt to navigate *specified browsing context* that has not yet [matured](#) (i.e., it has not passed the point of making its [Document](#) the [active document](#)), then cancel that attempt to navigate the [browsing context](#).
 2. If the *specified browsing context*'s [active document](#) is not the same [Document](#) as the [Document](#) of the *specified entry*, then run these substeps:
 1. [Prompt to unload](#) the [active document](#) of the *specified browsing context*. If the user [refused to allow the document to be unloaded](#), then abort these steps.
 2. [Unload](#) the [active document](#) of the *specified browsing context* with the [recycle](#) parameter set to false.
 3. [Traverse the history](#) of the *specified browsing context* to the *specified entry*.

When the user navigates through a [browsing context](#), e.g., using a browser's back and forward buttons, the user agent must [traverse the history by a delta](#) equivalent to the action specified by the user.



The `pushState()` method adds a state object entry to the history.

The `replaceState()` method updates the state object, title, and optionally the [URL](#) of the [current entry](#) in the history.

When either of these methods is invoked, the user agent must run the following steps:

1. If this [History](#) object is associated with a [Document](#) that is not [fully active](#), throw a "[SecurityError](#)" [DOMException](#).
2. Optionally, abort these steps. (For example, the user agent might disallow calls to these methods that are invoked on a timer, or from event listeners that are not triggered in response to a clear user action, or that are invoked in rapid succession.)
3. Let *targetRealm* be this [History](#) object's [relevant settings object](#)'s [Realm](#).
4. Let *cloned data* be a [StructuredClone](#)(*data*, *targetRealm*). Rethrow any exceptions.
5. If the third argument is not null, run these substeps:
 1. [Parse](#) the value of the third argument, relative to the [entry settings object](#).
 2. If that fails, throw a "[SecurityError](#)" [DOMException](#) and abort these steps.
 3. Let *new URL* be the [resulting URL record](#).
 4. Compare *new URL* to the document's [URL](#). If any component of these two [URL records](#) differ other than the [path](#), [query](#), and [fragment](#) components, then throw a "[SecurityError](#)" [DOMException](#) and abort these steps.
 5. If the [origin](#) of *new URL* is not the same as the [origin](#) of the [responsible document](#) specified by the [entry settings object](#), and either the [path](#) or [query](#) components of the two [URL records](#) compared in the previous step differ, throw a "[SecurityError](#)" [DOMException](#) and abort these steps. (This prevents sandboxed content from spoofing other pages on the same origin.)
6. If the third argument is null, then let *new URL* be the [URL](#) of the [current entry](#).
7. If the method invoked was the `pushState()` method:
 1. Remove all the entries in the [browsing context](#)'s [session history](#) after the [current entry](#). If the [current entry](#) is the

last entry in the session history, then no entries are removed.

NOTE:

This [doesn't necessarily have to affect](#) the user agent's user interface.

2. Remove any [tasks](#) queued by the [history traversal task source](#) that are associated with any [Document](#) objects in the [top-level browsing context's document family](#).
3. If appropriate, update the [current entry](#) to reflect any state that the user agent wishes to persist. The entry is then said to be [an entry with persisted user state](#).
4. Add a [state object](#) entry to the session history, after the [current entry](#), with *cloned data* as the [state object](#), the given *title* as the title, *new URL* as the [URL](#) of the entry, and the [scroll restoration mode](#) of the current entry in the [session history](#) as the scroll restoration mode.
5. Update the [current entry](#) to be this newly added entry.

Otherwise, if the method invoked was the [replaceState\(\)](#) method:

1. Update the [current entry](#) in the session history so that *cloned data* is the entry's new state object, the given *title* is the new title, and *new URL* is the entry's new [URL](#).
8. If the [current entry](#) in the session history represents a non-GET request (e.g., it was the result of a POST submission) then update it to instead represent a GET request.
9. Set the document's [URL](#) to *new URL*.

NOTE:

Since this is neither a [navigation](#) of the [browsing context](#) nor a [history traversal](#), it does not cause a hashchange event to be fired.

10. Let *targetRealm* be this [History](#) object's [relevant settings object's Realm](#).
11. Set [history.state](#) to [StructuredClone](#)(*cloned data*, *targetRealm*).
12. Let the [latest entry](#) of the [Document](#) of the [current entry](#) be the [current entry](#).

NOTE:

The *title* is purely advisory. User agents might use the title in the user interface.

User agents may limit the number of state objects added to the session history per page. If a page hits the user agent-defined limit, user agents must remove the entry immediately after the first entry for that [Document](#) object in the session history after having added the new entry. (Thus the state history acts as a FIFO buffer for eviction, but as a LIFO buffer for navigation.)

EXAMPLE 611

Consider a game where the user can navigate along a line, such that the user is always at some coordinate, and such that the user can bookmark the page corresponding to a particular coordinate, to return to it later.

A static page implementing the $x=5$ position in such a game could look like the following:

```
<!DOCTYPE HTML>
<!-- this is https://example.com/line?x=5 -->
<title>Line Game - 5</title>
<p>You are at coordinate 5 on the line.</p>
<p>
<a href="?x=6">Advance to 6</a> or
<a href="?x=4">retreat to 4</a>?
</p>
```

The problem with such a system is that each time the user clicks, the whole page has to be reloaded. Here instead is another way of doing it, using script:

```
<!DOCTYPE HTML>
<!-- this starts off as https://example.com/line?x=5 -->
<title>Line Game - 5</title>
<p>You are at coordinate <span>5</span> on the line.</p>
<p>
<a href="?x=6" onclick="go(1); return false;">Advance to 6</a> or
<a href="?x=4" onclick="go(-1); return false;">retreat to 4</a>?
</p>
<script>
var currentPage = 5; // prefilled by server
function go(d) {
  setupPage(currentPage + d);
  history.pushState(currentPage, document.title, '?x=' + currentPage);
}
onpopstate = function(event) {
  setupPage(event.state);
}
function setupPage(page) {
  currentPage = page;
  document.title = 'Line Game - ' + currentPage;
  document.getElementById('coord').textContent = currentPage;
  document.links[0].href = '?x=' + (currentPage+1);
  document.links[0].textContent = 'Advance to ' + (currentPage+1);
  document.links[1].href = '?x=' + (currentPage-1);
  document.links[1].textContent = 'retreat to ' + (currentPage-1);
}
</script>
```

In systems without script, this still works like the previous example. However, users that *do* have script support can now navigate much faster, since there is no network access for the same experience. Furthermore, contrary to the experience the user would have with just a naïve script-based approach, bookmarking and navigating the session history still work.

In the example above, the `data` argument to the `pushState()` method is the same information as would be sent to the server, but in a more convenient form, so that the script doesn't have to parse the URL each time the user navigates.

EXAMPLE 612

Applications might not use the same title for a [session history entry](#) as the value of the document's `<title>` element at that time. For example, here is a simple page that shows a block in the `<title>` element. Clearly, when navigating backwards to a previous state the user does not go back in time, and therefore it would be inappropriate to put the time in the session history title.

```
<!DOCTYPE HTML>
<TITLE>Line</TITLE>
<SCRIPT>
setInterval(function () { document.title = 'Line - ' + new Date(); }, 1000);
var i = 1;
function inc() {
  set(i+1);
  history.pushState(i, 'Line - ' + i);
}
function set(newI) {
  i = newI;
  document.forms.F.I.value = newI;
}
</SCRIPT>
<BODY ONPOPSTATE="set(event.state)">
<FORM NAME=F>
State: <OUTPUT NAME=I>1</OUTPUT> <INPUT VALUE="Increment" TYPE=BUTTON ONCLICK="inc()">
</FORM>
```

EXAMPLE 613

Most applications want to use the same [scroll restoration mode](#) value for all of their history entries. To achieve this they should set the `scrollRestoration` attribute as soon as possible (e.g., in the first `script` element in the document's `<head>` element) to ensure that any entry added to the history session gets the desired scroll restoration mode.

```
<head>
<script>
  if ('scrollRestoration' in history)
    history.scrollRestoration = 'manual';
</script>
</head>
```

§ 6.6.3. Implementation notes for session history

This section is non-normative.

The [History](#) interface is not meant to place restrictions on how implementations represent the session history to the user.

For example, session history could be implemented in a tree-like manner, with each page having multiple "forward" pages. This specification doesn't define how the linear list of pages in the [history](#) object are derived from the actual session history as seen from the user's perspective.

Similarly, a page containing two `<iframe>`s has a [history](#) object distinct from the `<iframe>`s' [history](#) objects, despite the fact that typical Web browsers present the user with just one "Back" button, with a session history that interleaves the navigation of the two inner frames and the outer page.

Security: It is suggested that to avoid letting a page "hijack" the history navigation facilities of a UA by abusing [pushState\(\)](#), the UA provide the user with a way to jump back to the previous page (rather than just going back to the previous state). For example, the back button could have a drop down showing just the pages in the session history, and not showing any of the states. Similarly, an aural browser could have two "back" commands, one that goes back to the previous state, and one that jumps straight back to the previous page.

For both [pushState\(\)](#) and [replaceState\(\)](#), user agents are encouraged to prevent abuse of these APIs via too-frequent calls or over-large state objects. As detailed above, the algorithm explicitly allows user agents to ignore any such calls when

appropriate.

§ 6.6.4. The `Location` interface

Each [Window](#) object is associated with a unique instance of a [Location](#) object, allocated when the [Window](#) object is created.

To create a [Location](#) object, run these steps:

1. Let *location* be a new [Location platform object](#).
2. Perform ! *location*.[[DefineOwnProperty]]("toString", { [[Value]]: [%ObjProto_toString%](#), [[Writable]]: false, [[Enumerable]]: false, [[Configurable]]: false }).
3. Perform ! *location*.[[DefineOwnProperty]]("toJSON", { [[Value]]: undefined, [[Writable]]: false, [[Enumerable]]: false, [[Configurable]]: false }).
4. Perform ! *location*.[[DefineOwnProperty]]("valueOf", { [[Value]]: [%ObjProto_valueOf%](#), [[Writable]]: false, [[Enumerable]]: false, [[Configurable]]: false }).
5. Perform ! *location*.[[DefineOwnProperty]]("@@toPrimitive", { [[Value]]: undefined, [[Writable]]: false, [[Enumerable]]: false, [[Configurable]]: false }).
6. Set the value of the [\[\[DefaultProperties\]\]](#) internal slot of *location* to *location*.[[OwnPropertyKeys]]().
7. Return *location*.

This definition is non-normative. Implementation requirements are given below this definition.

`document.location` [= *value*]

`window.location` [= *value*]

Returns a [Location](#) object with the current page's location.

Can be set, to navigate to another page.

The **`location`** attribute of the [Document](#) interface must return the [Location](#) object for that [Document](#) object's [global object](#), if it has a [browsing context](#), and null otherwise.

The **`location`** attribute of the [Window](#) interface must return the [Location](#) object for that [Window](#) object.

[Location](#) objects provide a representation of the [URL](#) of the [active document](#) of their [Document](#)'s [browsing context](#), and allow the [current entry](#) of the [browsing context](#)'s session history to be changed, by adding or replacing entries in the [history](#) object.

```
[Unforgeable]
interface Location {
  stringifier attribute USVString href;
  readonly attribute USVString origin;
  attribute USVString protocol;
  attribute USVString host;
  attribute USVString hostname;
  attribute USVString port;
  attribute USVString pathname;
  attribute USVString search;
  attribute USVString hash;

  void assign(USVString url);
  void replace(USVString url);
  void reload();

  [SameObject] readonly attribute USVString[] ancestorOrigins;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

`location.toString()`

`location.href`

Returns the [Location](#) object's URL.

Can be set, to navigate to the given URL.

`location.origin`

Returns the [Location](#) object's URL's origin.

`location.protocol`

Returns the [Location](#) object's URL's scheme.

Can be set, to navigate to the same URL with a changed scheme.

`location.host`

Returns the [Location](#) object's URL's host and port (if different from the default port for the scheme).

Can be set, to navigate to the same URL with a changed host and port.

`location.hostname`

Returns the [Location](#) object's URL's host.

Can be set, to navigate to the same URL with a changed host.

`location.port`

Returns the [Location](#) object's URL's port.

Can be set, to navigate to the same URL with a changed port.

`location.pathname`

Returns the [Location](#) object's URL's path.

Can be set, to navigate to the same URL with a changed path.

`location.search`

Returns the [Location](#) object's URL's query (includes leading "?" if non-empty).

Can be set, to navigate to the same URL with a changed query (ignores leading "?").

`location.hash`

Returns the [Location](#) object's URL's fragment (includes leading "#" if non-empty).

Can be set, to navigate to the same URL with a changed fragment (ignores leading "#").

`location.assign(url)`

Navigates to the given URL.

`location.replace(url)`

Removes the current page from the session history and navigates to the given URL.

`location.reload()`

Reloads the current page.

`location.ancestorOrigins`

Returns an array whose values are the origins of the ancestor [browsing contexts](#), from the [parent browsing context](#) to the [top-level browsing context](#).

A [Location](#) object has an associated **relevant Document**, which is this [Location](#) object's associated [Document](#) object's [browsing context](#)'s [active document](#).

A [Location](#) object has an associated **url**, which is this [Location](#) object's [relevant Document](#)'s [URL](#).

A [Location](#) object has an associated **ancestor origins array**. When a [Location](#) object is created, its [ancestor origins array](#) must be set to a array created from the list of strings that the following steps would produce:

1. Let *output* be an empty ordered list of strings.
2. Let *current* be the [browsing context](#) of the [Document](#) with which the [Location](#) object is associated.
3. *Loop*: If *current* has no [parent browsing context](#), jump to the step labeled *End*.
4. Let *current* be *current*'s [parent browsing context](#).
5. Append the [Unicode serialization](#) of *current*'s [active document](#)'s [origin](#) to *output* as a new value.
6. Return to the step labeled *Loop*.
7. *End*: Return *output*.

A [Location](#) object has an associated **Location-object-setter navigate** algorithm, which given a *url*, runs these steps:

1. If any of the following conditions are met, let *replacement flag* be unset; otherwise, let it be set:
 - This [Location](#) object's [relevant Document](#) has [completely loaded](#), or
 - In the [task](#) in which the algorithm is running, an [activation behavior](#) is currently being processed whose [click](#) event was [trusted](#), or
 - In the [task](#) in which the algorithm is running, the event listener for a [trusted](#) [click](#) event is being handled.
2. [Location-object navigate](#), given *url* and *replacement flag*.

To **Location-object navigate**, given a *url* and *replacement flag*, run these steps:

1. The [source browsing context](#) is the [responsible browsing context](#) specified by the [incumbent settings object](#).
2. [Navigate](#) the [browsing context](#) to *url*, with the [exceptions enabled flag](#) set. Rethrow any exceptions.

If the *replacement flag* is set or the [browsing context](#)'s [session history](#) contains only one [Document](#), and that was the about:blank [Document](#) created when the [browsing context](#) was created, then the navigation must be done with [replacement enabled](#).

The **href** attribute's getter must return this [Location](#) object's [URL](#), serialized.

The **href** attribute's setter must run these steps:

1. Let *newURL* be the [resulting URL string](#) of [parsing](#) the given value relative to the [entry settings object](#)'s [API base URL](#).
2. If that aborted with an error, throw a `TypeError` exception.
3. [Location-object-setter navigate](#) to *newURL*.

The **origin** attribute's getter must return the [Unicode serialization](#) of this [Location](#) object's [URL](#)'s [origin](#).

NOTE:

It returns the Unicode rather than the ASCII serialization for compatibility with `MessageEvent`.

The **protocol** attribute's getter must return this [Location](#) object's [URL](#)'s [scheme](#), followed by ":".

The **protocol** attribute's setter must run these steps:

1. Let *copyURL* be a copy of this [Location](#) object's [URL](#).
2. Let *possibleFailure* be the result of [basic URL parsing](#) the given value, followed by ":", with *copyURL* as *url* and [scheme start state](#) as *state override*.
3. If *possibleFailure* is failure, throw a `TypeError` exception.
4. If *copyURL*'s [scheme](#) is not "http" or "https", terminate these steps.
5. [Location-object-setter navigate](#) to *copyURL*.

The **host** attribute's getter must run these steps:

1. Let *url* be this [Location](#) object's [URL](#).
2. If *url*'s [host](#) is null, return the empty string.
3. If *url*'s [port](#) is null, return *url*'s [host](#), serialized.
4. Return *url*'s [host](#), serialized, followed by ":" and *url*'s [port](#), serialized.

The **host** attribute's setter must run these steps:

1. Let *copyURL* be a copy of this [Location](#) object's [URL](#).
2. If *copyURL*'s [non-relative flag](#) is set, terminate these steps.
3. [Basic URL parse](#) the given value, with *copyURL* as *url* and [host state](#) as *state override*.
4. [Location-object-setter navigate](#) to *copyURL*.

The **hostname** attribute's getter must run these steps:

1. If this [Location](#) object's [URL](#)'s [host](#) is null, return the empty string.
2. Return this [Location](#) object's [URL](#)'s [host](#), serialized.

The **hostname** attribute's setter must run these steps:

1. Let *copyURL* be a copy of this [Location](#) object's [URL](#).
2. If *copyURL*'s [non-relative flag](#) is set, terminate these steps.
3. [Basic URL parse](#) the given value, with *copyURL* as *url* and [hostname state](#) as *state override*.
4. [Location-object-setter navigate](#) to *copyURL*.

The **port** attribute's getter must run these steps:

1. If this [Location](#) object's [URL](#)'s [port](#) is null, return the empty string.
2. Return this [Location](#) object's [URL](#)'s [port](#), serialized.

The **port** attribute's setter must run these steps:

1. Let *copyURL* be a copy of this [Location](#) object's [URL](#).
2. If *copyURL*'s [host](#) is null, *copyURL*'s [non-relative flag](#) is set, or *copyURL*'s [scheme](#) is "file", terminate these steps.
3. [Basic URL parse](#) the given value, with *copyURL* as *url* and [port state](#) as *state override*.
4. [Location-object-setter navigate](#) to *copyURL*.

The **pathname** attribute's getter must run these steps:

1. Let *url* be this [Location](#) object's [URL](#).
2. If *url*'s [non-relative flag](#) is set, return the first string in *url*'s [path](#).
3. Return "/", followed by the strings in *url*'s [path](#) (including empty strings), separated from each other by "/".

The **pathname** attribute's setter must run these steps:

1. Let *copyURL* be a copy of this [Location](#) object's [URL](#).
2. If *copyURL*'s [non-relative flag](#) is set, terminate these steps.
3. Set *copyURL*'s [path](#) to the empty list.
4. [Basic URL parse](#) the given value, with *copyURL* as *url* and [path start state](#) as *state override*.

5. [Location-object-setter navigate](#) to *copyURL*.

The **search** attribute's getter must run these steps:

1. If this [Location](#) object's [URL](#)'s [query](#) is either null or the empty string, return the empty string.
2. Return "?", followed by this [Location](#) object's [URL](#)'s [query](#).

The **search** attribute's setter must run these steps:

1. Let *copyURL* be a copy of this [Location](#) object's [URL](#).
2. If the given value is the empty string, set *copyURL*'s [query](#) to null.
3. Otherwise, run these substeps:
 1. Let *input* be the given value with a single leading "?" removed, if any.
 2. Set *copyURL*'s [query](#) to the empty string.
 3. [Basic URL parse](#) *input*, with *copyURL* as *url* and [query state](#) as *state override*, and the [relevant Document](#)'s [document's character encoding](#) as *encoding override*.
4. [Location-object-setter navigate](#) to *copyURL*.

The **hash** attribute's getter must run these steps:

1. If this [Location](#) object's [URL](#)'s [fragment](#) is either null or the empty string, return the empty string.
2. Return "#", followed by this [Location](#) object's [URL](#)'s [fragment](#).

The **hash** attribute's setter must run these steps:

1. Let *copyURL* be a copy of this [Location](#) object's [URL](#).
2. If *copyURL*'s [scheme](#) is "javascript", terminate these steps.
3. If the given value is the empty string, set *copyURL*'s [fragment](#) to null.
4. Otherwise, run these substeps:
 1. Let *input* be the given value with a single leading "#" removed, if any.
 2. Set *copyURL*'s [fragment](#) to the empty string.
 3. [Basic URL parse](#) *input*, with *copyURL* as *url* and [fragment state](#) as *state override*.
5. [Location-object-setter navigate](#) to *copyURL*.



When the **assign(*url*)** method is invoked, the user agent must run the following steps:

1. [Parse](#) *url*, relative to the [API base URL](#) specified by the [entry settings object](#) and let *parsedURL* be the [resulting URL string](#).
If this is not successful, throw a "SyntaxError" [DOMException](#) and abort these steps.
2. [Location-object navigate](#) to *parsedURL*.

When the **replace(*url*)** method is invoked, the user agent must run the following steps:

1. [Parse](#) *url*, relative to the [API base URL](#) specified by the [entry settings object](#) and let *parsedURL* be the [resulting URL string](#).
If this is not successful, throw a "SyntaxError" [DOMException](#) and abort these steps.
2. [Location-object navigate](#) to *parsedURL* with the *replacement flag* set.

When the **reload()** method is invoked, the user agent must run the appropriate steps from the following list:

↪ If the currently executing [task](#) is the dispatch of a [resize event](#) in response to the user resizing the [browsing context](#)

Repaint the [browsing context](#) and abort these steps.

↪ If the [browsing context](#)'s [active document](#) is an [iframe srcdoc document](#)

Reprocess the [iframe attributes](#) of the [browsing context](#)'s [browsing context container](#).

↪ If the [browsing context](#)'s [active document](#) has its [reload override flag](#) set

Perform an [overridden reload](#), with the [browsing context](#) being navigated as the [responsible browsing context](#).

↪ Otherwise

Navigate the [browsing context](#) to the document's [URL](#), with the [exceptions enabled flag](#) set and [replacement enabled](#). The [source browsing context](#) must be the [browsing context](#) being navigated. This is a [reload-triggered navigation](#). Rethrow any exceptions.

When a user requests that the [active document](#) of a [browsing context](#) be reloaded through a user interface element, the user agent should [navigate](#) the [browsing context](#) to the same resource as that [Document](#), with [replacement enabled](#). In the case of non-idempotent methods (e.g., HTTP POST), the user agent should prompt the user to confirm the operation first, since otherwise transactions (e.g., purchases or database modifications) could be repeated. User agents may allow the user to explicitly override any caches when reloading. If [browsing context](#)'s [active document](#)'s [reload override flag](#) is set, then the user agent may instead perform an [overridden reload](#) rather than the navigation described in this paragraph (with the [browsing context](#) being reloaded as the [source browsing context](#)).

The [ancestorOrigins](#) attribute's getter must run these steps:

1. If this [Location](#) object's [relevant Document](#)'s [origin](#) is not [same origin-domain](#) with the [entry settings object](#)'s [origin](#), then throw a ["SecurityError" DOMException](#).
2. Otherwise, return this [Location](#) object's [ancestor origins array](#).

The [Location](#) object requires additional logic beyond IDL for security purposes. The internal slot and internal methods [Location](#) objects must implement are defined below.

Every [Location](#) object has a [\[\[DefaultProperties\]\]](#) internal slot representing its own properties at time of its creation.

§ 6.7. Browsing the Web

§ 6.7.1. Navigating across documents

Certain actions cause the [browsing context](#) to [navigate](#) to a new resource. A user agent may provide various ways for the user to explicitly cause a browsing context to navigate, in addition to those defined in this specification.

EXAMPLE 614

For example, [following a hyperlink](#), [§4.10.22 Form submission](#), and the `window.open()` and `location.assign()` methods can all cause a browsing context to navigate.

NOTE:

A resource has a URL, but that might not be the only information necessary to identify it. For example, a [form submission](#) that uses HTTP POST would also have the HTTP method and payload. Similarly, an [iframe srcdoc document](#) needs to know the data it is to use.

Navigation always involves **source browsing context**, which is the browsing context which was responsible for starting the navigation.

When a browsing context is **navigated** to a new resource, the user agent must run the following steps:

1. If the [source browsing context](#) is not [allowed to navigate](#) the [browsing context](#) being navigated, then abort these steps.

If these steps are aborted here, the user agent may instead offer to open the new resource in a new [top-level browsing context](#) or in the [top-level browsing context](#) of the [source browsing context](#), at the user's option, in which case the user

agent must [navigate](#) that designated [top-level browsing context](#) to the new resource as if the user had requested it independently.

NOTE:

Doing so, however, can be dangerous, as it means that the user is overriding the author's explicit request to sandbox the content.

If the [navigate](#) algorithm was invoked optionally with an **exceptions enabled flag**, and it is aborted on this step, then in addition to aborting this algorithm, the user agent must also throw a ["SecurityError" DOMException](#).

2. If there is a preexisting attempt to navigate the [browsing context](#), and the [source browsing context](#) is the same as the [browsing context](#) being navigated, and that attempt is currently running the [unload a document](#) algorithm, and the [origin](#) of the [URL](#) of the resource being loaded in that navigation is not the [same origin](#) as the [origin](#) of the [URL](#) of the resource being loaded in *this* navigation, then abort these steps without affecting the preexisting attempt to navigate the [browsing context](#).
3. If a [task](#) queued by the [traverse the history by a delta](#) algorithm is running the [unload a document](#) algorithm for the [active document](#) of the [browsing context](#) being navigated, then abort these steps without affecting the [unload a document](#) algorithm or the aforementioned history traversal task.
4. If the [prompt to unload a document](#) algorithm is being run for the [active document](#) of the [browsing context](#) being navigated, then abort these steps without affecting the [prompt to unload a document](#) algorithm.
5. Let *gone async* be false.

NOTE:

The handle redirects step later in this algorithm can in certain cases jump back to the step labeled [fragment identifiers](#). Since, between those two steps, this algorithm goes from operating immediately in the context of the calling [task](#) to operating [in parallel](#) independent of the [event loop](#), some of the intervening steps need to be able to handle both being run as part of a [task](#) and running [in parallel](#). The *gone async* flag is thus used to make these steps aware of which mode they are operating in.

6. *Fragment identifiers*: If this is not a **reload-triggered navigation**: apply the [URL parser](#) algorithm to the [absolute URL](#) of the new resource and the [address](#) of the [active document](#) of the [browsing context](#) being navigated; if all the components of the resulting [parsed URLs](#), ignoring any [fragment](#) components, are identical, and the new resource is to be fetched using GET, and the [URL record](#) of the new resource has a [fragment](#) component that is not null (even if it is empty), then [navigate to that fragment identifier](#) and abort these steps.
7. If *gone async* is false, cancel any preexisting but not yet [mature](#) attempt to navigate the [browsing context](#), including canceling any instances of the [fetch](#) algorithm started by those attempts. If one of those attempts has already created and initialized a new [Document](#) object, [abort](#) that [Document](#) also. (Navigation attempts that have [matured](#) already have session history entries, and are therefore handled during the [update the session history with the new page](#) algorithm, later.)
8. If the new resource is to be handled using a mechanism that does not affect the browsing context, e.g., ignoring the navigation request altogether because the specified scheme is not one of the supported protocols, then abort these steps and [proceed with that mechanism instead](#).
9. If *gone async* is false, [prompt to unload](#) the [Document](#) object. If the user [refused to allow the document to be unloaded](#), then abort these steps.

If this instance of the [navigation](#) algorithm gets canceled while this step is running, the [prompt to unload a document](#) algorithm must nonetheless be run to completion.

10. If *gone async* is false, [abort](#) the [active document](#) of the [browsing context](#).
11. If the new resource is to be handled by displaying some sort of inline content, e.g., an error message because the specified scheme is not one of the supported protocols, or an inline prompt to allow the user to select a [registered handler](#) for the given scheme, then [display the inline content](#) and abort these steps.

NOTE:

In the case of a registered handler being used, the algorithm will be reinvoked with a new URL to handle the request.

12. If the [browsing context](#) being navigated is a [nested browsing context](#), then put it in the [delaying load events mode](#).

The user agent must take this [nested browsing context](#) out of the [delaying load events mode](#) when this [navigation](#) algorithm later matures, or when it terminates (whether due to having run all the steps, or being canceled, or being aborted), whichever happens first.

13. This is the step that attempts to obtain the resource, if necessary. Jump to the first appropriate substep:

If the resource has already been obtained (e.g., because it is being used to populate an [<object>](#) element's new [child browsing context](#))

Skip this step. The data is already available.

If the new resource is a [URL](#) whose [scheme](#) is javascript

[Queue a task](#) to run these "javascript: URL" steps, associated with the [active document](#) of the [browsing context](#) being navigated:

1. If the [origin](#) of the [source browsing context](#) is not the [same origin](#) as the [origin](#) of the [active document](#) of the [browsing context](#) being navigated, then let *result* be undefined, and jump to the step labeled *process results* below.
2. Let *urlRecord* be the result of running the [URL parser](#) on the [URL](#) of the new resource.
3. Let *script source* be the empty string.
4. Append the first string of *urlRecord*'s [path](#) component to *script source*.
5. If *urlRecord*'s [query](#) component is not null, then first append a U+003F QUESTION MARK character (?) to *script source*, and then append *urlRecord*'s [query](#) component to *script source*.
6. If *urlRecord*'s [fragment](#) component is not null, then first append a U+0023 NUMBER SIGN character (#) to *script source*, and then append *urlRecord*'s [fragment](#) component to *script source*.
7. Replace *script source* with the result of applying the [percent decode](#) algorithm to *script source*.
8. Replace *script source* with the result of applying the [UTF-8 decode](#) algorithm to *script source*.
9. Let *address* be the [address](#) of the [active document](#) of the [browsing context](#) being navigated.
10. Let *settings* be the [relevant settings object](#) of the [browsing context](#) being navigated.
11. Let *script* be the result of [creating a classic script](#) given *script source* and *settings*.
12. Let *result* be the result of [running the classic script](#) *script*. If evaluation was unsuccessful, let *result* be undefined instead. (The result will also be undefined if [scripting is disabled](#).)
13. *Process results*: If [Type](#)(*result*) is not String, then the result of obtaining the resource for the URL is a [response](#) whose [status](#) is 204.

Otherwise, the result of obtaining the resource for the URL is a [response](#) whose [header list](#) consists of Content-Type/[text/html](#) and whose [body](#) is *result*, and whose [HTTPS state](#) is *settings*'s [HTTPS state](#).

When it comes time to [set the document's address](#) in the [navigation algorithm](#), use *address* as the [override URL](#).

The [task source](#) for this [task](#) is the [DOM manipulation task source](#).

EXAMPLE 615

So for example a `javascript: URL` in an `href` attribute of an `<a>` element would only be evaluated when the link was followed, while such a URL in the `src` attribute of an `<iframe>` element would be evaluated in the context of the `iframe`'s own [nested browsing context](#) when the `iframe` is being set up; once evaluated, its return value (if it was not void) would replace that [browsing context](#)'s [Document](#), thus also changing the `Window` object of that [browsing context](#).

Otherwise

1. Let *request* be the new resource.
 2. If *request* is a URL, set *request* to a new [request](#) whose [URL](#) is *request*.
 3. Set *request*'s [client](#) to the [source browsing context](#)'s [active document](#)'s `Window` object's [environment settings object](#), [target browsing context](#) to the [browsing context](#) being navigated, [destination](#) to "document", mode to "navigate", [credentials mode](#) to "include", [use-URL-credentials flag](#), and [redirect mode](#) to "manual".
 4. Set *request*'s [omit-Origin-header flag](#).
 5. If *request*'s [method](#) is not GET, or, if the [navigation algorithm](#) was invoked as a result of the [form submission algorithm](#), then if there is an [origin](#) of the [active document](#) of the [source browsing context](#), unset *request*'s [omit-Origin-header flag](#).
 6. Otherwise, if the [browsing context](#) being navigated is a [child browsing context](#), and the [browsing context container](#) of the [browsing context](#) being navigated has a [browsing context scope origin](#), set *request*'s [origin](#) to that [browsing context scope origin](#) and unset *request*'s [omit-Origin-header flag](#).
 7. [Fetch](#) *request*.
14. If *gone async* is false, return to whatever algorithm invoked the navigation steps and continue running these steps [in parallel](#).
15. Let *gone async* be true.
16. Wait for one or more bytes to be available or for the user agent to establish that the resource in question is empty.
During this time, the user agent may allow the user to cancel this navigation attempt or start other navigation attempts.
17. *Handle redirects*: If fetching the resource results in a redirect, and either the [URL](#) of the target of the redirect has the [same origin](#) as the original resource, or the resource is being obtained using the POST method or a safe method (in HTTP terms), return to [the step labeled fragment identifiers](#) with the new resource, except that if the [URL](#) of the target of the redirect does not have a fragment identifier and the [URL](#) of the resource that led to the redirect does, then the fragment identifier of the resource that led to the redirect must be propagated to the [URL](#) of the target of the redirect.

EXAMPLE 616

So for instance, if the original URL was "https://example.com/#!sample" and "https://example.com/" is found to redirect to "https://example.com/", the URL of the new resource will be "https://example.com/#!sample".

Otherwise, if fetching the resource results in a redirect but the [URL](#) of the target of the redirect does not have the [same origin](#) as the original resource and the resource is being obtained using a method that is neither the POST method nor a safe method (in HTTP terms), then abort these steps. The user agent may indicate to the user that the navigation has been aborted for security reasons.

18. *Resource handling*: If the resource's out-of-band metadata (e.g., HTTP headers), not counting any [type information](#) (such as the Content-Type HTTP header), requires some sort of processing that will not affect the browsing context, then perform that processing and abort these steps.

Such processing might be triggered by, amongst other things, the following:

- HTTP status codes (e.g., 204 No Content or 205 Reset Content)
- Network errors (e.g., the network interface being unavailable)
- Cryptographic protocol failures (e.g., an incorrect TLS certificate)

Responses with HTTP Content-Disposition headers specifying the attachment disposition type must be handled [as a download](#).

HTTP 401 responses that do not include a challenge recognized by the user agent must be processed as if they had no challenge, e.g., rendering the entity body as if the response had been 200 OK.

User agents may show the entity body of an HTTP 401 response even when the response does include a recognized challenge, with the option to login being included in a non-modal fashion, to enable the information provided by the server to be used by the user before authenticating. Similarly, user agents should allow the user to authenticate (in a non-modal fashion) against authentication challenges included in other responses such as HTTP 200 OK responses, effectively allowing resources to present HTTP login forms without requiring their use.

19. Let *type* be the computed type of the resource.
20. If the user agent has been configured to process resources of the given *type* using some mechanism other than rendering the content in a [browsing context](#), then skip this step. Otherwise, if the *type* is one of the following types, jump to the appropriate entry in the following list, and process the resource as described there:

↪ **an [HTML MIME type](#)**

Follow the steps given in the [HTML document](#) section, and then, once they have completed, abort this [navigate](#) algorithm.

↪ **an [XML MIME type](#) that is not an [explicitly supported XML type](#)**

Follow the steps given in the [XML document](#) section. If that section determines that the content is *not* to be displayed as a generic XML document, then proceed to the next step in this overall set of steps. Otherwise, once the steps given in the [XML document](#) section have completed, abort this [navigate](#) algorithm.

↪ **a [JavaScript MIME type](#)**

↪ **a [JSON MIME type](#) that is not an [explicitly supported JSON type](#)**

↪ **"text/css"**

↪ **"text/plain"**

↪ **"text/vtt"**

Follow the steps given in the [plain text file](#) section, and then, once they have completed, abort this [navigate](#) algorithm.

↪ **"multipart/x-mixed-replace"**

Follow the steps given in the [§12.2 multipart/x-mixed-replace](#) section, and then, once they have completed, abort this [navigate](#) algorithm.

↪ **A supported image, video, or audio type**

Follow the steps given in the [media](#) section, and then, once they have completed, abort this [navigate](#) algorithm.

↪ **A type that will use an external application to render the content in the [browsing context](#)**

Follow the steps given in the [plugin](#) section, and then, once they have completed, abort this [navigate](#) algorithm.

An **explicitly supported XML type** is one for which the user agent is configured to use an external application to render the content (either a [plugin](#) rendering directly in the [browsing context](#), or a separate application), or one for which the user agent has dedicated processing rules (e.g., a Web browser with a built-in Atom feed viewer would be said to explicitly support the application/atom+xml MIME type).

The term **JSON MIME type** is used to refer to the [MIME types](#) application/json, text/json, and any [MIME type](#) whose subtype ends with the five characters "+json".

An **explicitly supported JSON type** is one for which the user agent is configured to use an external application to render the content (either a [plugin](#) rendering directly in the [browsing context](#), or a separate application), or one for which the user agent has dedicated processing rules.

Setting the document's address: If there is no **override URL**, then any [Document](#) created by these steps must have its [address](#) set to the [URL](#) that was originally to be fetched, ignoring any other data that was used to obtain the resource.

However, if there is an [override URL](#), then any [Document](#) created by these steps must have its [address](#) set to that [URL](#) instead.

NOTE:

An [override URL](#) is set when [dereferencing a javascript: URL](#) and when performing [an overridden reload](#).

Initializing a new Document object: when a [Document](#) is created as part of the above steps, the user agent will be required to additionally run the following algorithm after creating the new object:

1. If *browsingContext*'s only entry in its [session history](#) is the [about:blank Document](#) that was added when *browsingContext* was [created](#), and navigation is occurring with [replacement enabled](#), and that [Document](#) has the [same origin](#) as the new [Document](#), then

1. Let *window* be the [Window](#) object of that [Document](#).
2. Change the [document](#) attribute of *window* to point to the new [Document](#).

2. Otherwise,

1. Call the JavaScript [InitializeHostDefinedRealm\(\)](#) abstract operation with the following customizations:

- For the global object, create a new [Window](#) object *window*.
- For the global *this* value, use *browsingContext*'s [WindowProxy](#) object.
- Let *realm execution context* be the created [JavaScript execution context](#).
- Do not obtain any source texts for scripts.

2. [Set up a browsing context environment settings object](#) with *realm execution context*, and let *settings object* be the result.

3. Set *window*'s [associated Document](#) to the new [Document](#).

3. Set *browsingContext*'s [WindowProxy](#) object's `[[Window]]` internal slot value to *window*.

4. Set the [Document](#)'s [HTTPS state](#) to the [HTTPS state](#) of the resource used to generate the document.

5. Execute the [Initialize a Document's CSP list](#) algorithm on the [Document](#) object and the resource used to generate the document. [\[CSP3\]](#)

6. Set [the document's referrer](#) to the *address of the resource from which Request-URLs are obtained* as determined when the fetch algorithm obtained the resource, if that algorithm was used and determined such a value; otherwise, set it to the empty string.

7. [Implement the sandboxing](#) for the [Document](#).

8. If the [active sandboxing flag set](#) of the [Document](#)'s [browsing context](#) or any of its [ancestor browsing contexts](#) (if any) have the [sandboxed fullscreen browsing context flag](#) set, then skip this step.

If the [Document](#)'s [browsing context](#) has a [browsing context container](#) and either it is not an `<iframe>` element, or it does not have the `allowfullscreen` attribute specified, or its [Document](#) does not have the [fullscreen enabled flag](#) set, then also skip this step.

Otherwise, set the [Document](#)'s [fullscreen enabled flag](#).

9. *Non-document content:* If, given *type*, the new resource is to be handled by displaying some sort of inline content, e.g., a native rendering of the content, an error message because the specified type is not supported, or an inline prompt to allow the user to select [a registered handler](#) for the given type, then [display the inline content](#), and then abort these steps.

NOTE:

In the case of a registered handler being used, the algorithm will be reinvoked with a new URL to handle the request.

10. Otherwise, the document's *type* is such that the resource will not affect the browsing context, e.g., because the resource is to be handed to an external application or because it is an unknown type that will be processed [as a download](#). [Process the resource appropriately](#).

When a resource is handled by **passing its URL or data to an external software package** separate from the user agent (e.g., handing a `mailto:` URL to a mail client, or a Word document to a word processor), user agents should attempt to mitigate the risk that this is an attempt to exploit the target software, e.g., by prompting the user to confirm that the [source browsing context's active document's origin](#) is to be allowed to invoke the specified software. In particular, if the [navigate](#) algorithm, when it was invoked, was not [allowed to show a popup](#), the user agent should not invoke the external software package without prior user confirmation.

EXAMPLE 617

For example, there could be a vulnerability in the target software's URL handler which a hostile page would attempt to exploit by tricking a user into clicking a link.



Some of the sections below, to which the above algorithm defers in certain cases, require the user agent to **update the session history with the new page**. When a user agent is required to do this, it must [queue a task](#) (associated with the [Document](#) object of the [current entry](#), not the new one) to run the following steps:

1. [Unload](#) the [Document](#) object of the [current entry](#), with the *recycle* parameter set to false.

If this instance of the [navigate](#) algorithm is canceled while this step is running the [unload a document](#) algorithm, then the [unload a document](#) algorithm must be allowed to run to completion, but this instance of the [navigate](#) algorithm must not run beyond this step. (In particular, for instance, the cancelation of this algorithm does not abort any event dispatch or script execution occurring as part of unloading the document or its descendants.)

2. **If the navigation was initiated for *entry update* of an entry**

1. Replace the [Document](#) of the entry being updated, and any other entries that referenced the same document as that entry, with the new [Document](#).
2. [Traverse the history](#) to the new entry.

NOTE:

This can only happen if the entry being updated is not the [current entry](#), and can never happen with [replacement enabled](#). (It happens when the user tried to traverse to a session history entry that no longer had a [Document](#) object.)

Otherwise

1. Remove all the entries in the [browsing context's session history](#) after the [current entry](#). If the [current entry](#) is the last entry in the session history, then no entries are removed.

NOTE:

This [doesn't necessarily have to affect](#) the user agent's user interface.

2. Append a new entry at the end of the History object representing the new resource and its [Document](#) object, related state, and the default [scroll restoration mode](#) of "auto".
3. [Traverse the history](#) to the new entry. If the navigation was initiated with [replacement enabled](#), then the traversal must itself be initiated with [replacement enabled](#).
3. The [navigate algorithm](#) has now **matured**.
4. *Fragment identifier loop*: [Spin the event loop](#) for a user-agent-defined amount of time, as desired by the user agent implementor. (This is intended to allow the user agent to optimize the user experience in the face of performance concerns.)
5. If the [Document](#) object has no parser, or its parser has [stopped parsing](#), or the user agent has reason to believe the user

is no longer interested in scrolling to the fragment identifier, then abort these steps.

6. Scroll to the [fragment identifier](#) given in [the document's address](#). If this fails to find [an indicated part of the document](#), then return to the *fragment identifier loop* step.

The [task source](#) for this [task](#) is the [networking task source](#).

§ 6.7.2. Page load processing model for HTML files

When an HTML document is to be loaded in a [browsing context](#), the user agent must [queue a task](#) to create a [Document](#) object, mark it as being an [HTML document](#), set its [content type](#) to "[text/html](#)", initialize the [Document](#) object, and finally create an [HTML parser](#) and associate it with the [Document](#). Each [task](#) that the [networking task source](#) places on the [task queue](#) while fetching runs must then fill the parser's [input byte stream](#) with the fetched bytes and cause the [HTML parser](#) to perform the appropriate processing of the input stream.

NOTE:

The [input byte stream](#) converts bytes into characters for use in the [tokenizer](#). This process relies, in part, on character encoding information found in the real [Content-Type metadata](#) of the resource; the "computed type" is not used for this purpose.

When no more bytes are available, the user agent must [queue a task](#) for the parser to process the implied EOF character, which eventually causes a load event to be fired.

After creating the [Document](#) object, but before any script execution, certainly before the parser [stops](#), the user agent must [update the session history with the new page](#).

The [task source](#) for the two tasks mentioned in this section must be the [networking task source](#).

§ 6.7.3. Page load processing model for XML files

When faced with displaying an XML file inline, user agents must follow the requirements defined in the XML and Namespaces in XML recommendations, RFC 7303, DOM, and other relevant specifications to create a [Document](#) object and a corresponding [XML parser](#). [\[XML\]](#) [\[XML-NAMES\]](#) [\[RFC7303\]](#) [\[DOM\]](#)

NOTE:

At the time of writing, the XML specification community had not actually yet specified how XML and the DOM interact.

After the [Document](#) is created, the user agent must initialize the [Document](#) object.

The actual HTTP headers and other metadata, not the headers as mutated or implied by the algorithms given in this specification, are the ones that must be used when determining the character encoding according to the rules given in the above specifications. Once the character encoding is established, the [document's character encoding](#) must be set to that character encoding.

User agents may examine the namespace of the root [Element](#) node of this [Document](#) object to perform namespace-based dispatch to alternative processing tools, e.g., determining that the content is actually a syndication feed and passing it to a feed handler. If such processing is to take place, abort the steps in this section, and jump to the next step (labeled *non-document content*) in the [navigate](#) steps above.

Otherwise, then, with the newly created [Document](#), the user agent must [update the session history with the new page](#). User agents may do this before the complete document has been parsed (thus achieving *incremental rendering*), and must do this before any scripts are to be executed.

Error messages from the parse process (e.g., XML namespace well-formedness errors) may be reported inline by mutating the [Document](#).

§ 6.7.4. Page load processing model for text files

When a plain text document is to be loaded in a [browsing context](#), the user agent must [queue a task](#) to create a [Document](#) object, mark it as being an [HTML document](#), set its [content type](#) to the [computed MIME type](#) of the resource (*type* in the [navigate](#) algorithm), initialize the [Document](#) object, create an [HTML parser](#), associate it with the [Document](#), act as if the tokenizer had emitted a start tag token with the tag name "pre" followed by a single U+000A LINE FEED (LF) character, and switch the [HTML parser](#)'s tokenizer to the [§8.2.4.7 PLAINTEXT state](#). Each [task](#) that the [networking task source](#) places on the [task queue](#) while fetching runs must then fill the parser's [input byte stream](#) with the fetched bytes and cause the [HTML parser](#) to perform the appropriate processing of the input stream.

The rules for how to convert the bytes of the plain text document into actual characters, and the rules for actually rendering the text to the user, are defined by the specifications for the [computed MIME type](#) of the resource (*type* in the [navigate](#) algorithm).

The [document's character encoding](#) must be set to the character encoding used to decode the document.

When no more bytes are available, the user agent must [queue a task](#) for the parser to process the implied EOF character, which eventually causes a load event to be fired.

After creating the [Document](#) object, but potentially before the page has finished parsing, the user agent must [update the session history with the new page](#).

User agents may add content to the [<head>](#) element of the [Document](#), e.g., to link to a style sheet, provide a script, give the document a [<title>](#), etc.

NOTE:

In particular, if the user agent supports the Format=Flowed feature of RFC3676 then the user agent would need to apply extra styling to cause the text to wrap correctly and to handle the quoting feature. [\[RFC3676\]](#)

The [task source](#) for the two tasks mentioned in this section must be the [networking task source](#).

§ 6.7.5. Page load processing model for multipart/x-mixed-replace resources

When a resource with the type `multipart/x-mixed-replace` is to be loaded in a [browsing context](#), the user agent must parse the resource using the rules for multipart types. [\[RFC2046\]](#)

For each body part obtained from the resource, the user agent must run a new instance of the [navigate](#) algorithm, starting from the *resource handling* step, using the new body part as the resource being navigated, with [replacement enabled](#) if a previous body part from the same resource resulted in a [Document](#) object being created and [initialized](#), and otherwise using the same setup as the [navigate](#) attempt that caused this section to be invoked in the first place.

For the purposes of algorithms processing these body parts as if they were complete stand-alone resources, the user agent must act as if there were no more bytes for those resources whenever the boundary following the body part is reached.

NOTE:

Thus, load events (and for that matter unload events) do fire for each body part loaded.

§ 6.7.6. Page load processing model for media

When an image, video, or audio resource is to be loaded in a [browsing context](#), the user agent should create a [Document](#) object, mark it as being an [HTML document](#), set its [content type](#) to the [computed MIME type](#) of the resource (*type* in the [navigate](#) algorithm), initialize the [Document](#) object, append an [<html>](#) element to the [Document](#), append a [<head>](#) element and a [<body>](#) element to the [<html>](#) element, append an element *host element* for the media, as described below, to the [<body>](#) element, and set the appropriate attribute of the element *host element*, as described below, to the address of the image, video, or audio resource.

The element *host element* to create for the media is the element given in the table below in the second cell of the row whose first cell describes the media. The appropriate attribute to set is the one given by the third cell in that same row.

Type of media	Element for the media	Appropriate attribute
Image	<code>img</code>	<code>src</code>

Video	video	src
Audio	audio	src

Then, the user agent must act as if it had [stopped parsing](#).

After creating the Document object, but potentially before the page has finished fully loading, the user agent must [update the session history with the new page](#).

User agents may add content to the [<head>](#) element of the [Document](#), or attributes to the element *host element*, e.g., to link to a style sheet, provide a script, give the document a [<title>](#), make the media [autoplay](#), etc.

§ 6.7.7. Page load processing model for content that uses plugins

When a resource that requires an external resource to be rendered is to be loaded in a [browsing context](#), the user agent should create a [Document](#) object, mark it as being an [HTML document](#) and mark it as being a **plugin document**, set its [content type](#) to the [computed MIME type](#) of the resource (*type* in the [navigate](#) algorithm), initialize the [Document](#) object, append an [<html>](#) element to the [Document](#), append a [<head>](#) element and a [<body>](#) element to the [<html>](#) element, append an embed to the [<body>](#) element, and set the `src` attribute of the [<embed>](#) element to the address of the resource.

NOTE:

The term [plugin document](#) is used by *Content Security Policy* as part of the mechanism that ensures iframes can't be used to evade plugin-types directives. [\[CSP3\]](#)

Then, the user agent must act as if it had [stopped parsing](#).

After creating the Document object, but potentially before the page has finished fully loading, the user agent must [update the session history with the new page](#).

User agents may add content to the [<head>](#) element of the [Document](#), or attributes to the [<embed>](#) element, e.g., to link to a style sheet, to give the document a [<title>](#), etc.

NOTE:

If the [Document's active sandboxing flag set](#) has its [sandboxed plugins browsing context flag set](#), the synthesized [<embed>](#) element will fail to render the content if the relevant [plugin](#) cannot be [secured](#).

§ 6.7.8. Page load processing model for inline content that doesn't have a DOM

When the user agent is to display a user agent page inline in a [browsing context](#), the user agent should create a [Document](#) object, mark it as being an [HTML document](#), set its [content type](#) to "[text/html](#)", initialize the [Document](#) object, and then either associate that [Document](#) with a custom rendering that is not rendered using the normal [Document](#) rendering rules, or mutate that [Document](#) until it represents the content the user agent wants to render.

Once the page has been set up, the user agent must act as if it had [stopped parsing](#).

After creating the Document object, but potentially before the page has been completely set up, the user agent must [update the session history with the new page](#).

§ 6.7.9. Navigating to a fragment identifier

When a user agent is supposed to navigate to a fragment identifier, then the user agent must run the following steps:

1. Remove all the entries in the [browsing context's session history](#) after the [current entry](#). If the [current entry](#) is the last entry in the session history, then no entries are removed.

NOTE:

This [doesn't necessarily have to affect](#) the user agent's user interface.

2. Remove any [tasks](#) queued by the [history traversal task source](#) that are associated with any [Document](#) objects in the

[top-level browsing context's document family](#).

3. Append a new entry at the end of the History object representing the new resource and its [Document](#) object, related state, and current history scroll restoration preference. Its [URL](#) must be set to the address to which the user agent was [navigating](#). The title must be left unset.
4. [Traverse the history](#) to the new entry, with the *non-blocking events* flag set. This will [scroll to the fragment identifier](#) given in what is now [the document's address](#).

NOTE:

If the scrolling fails because the relevant [ID](#) has not yet been parsed, then the original [navigation](#) algorithm will take care of the scrolling instead, as the last few steps of its [update the session history with the new page](#) algorithm.



When the user agent is required to **scroll to the fragment identifier** and [the indicated part of the document](#), if any, is [being rendered](#), the user agent must either change the scrolling position of the document using the following algorithm, or perform some other action such that [the indicated part of the document](#) is brought to the user's attention. If there is no indicated part, or if the indicated part is not [being rendered](#), then the user agent must do nothing. The aforementioned algorithm is as follows:

1. Let *target* be [the indicated part of the document](#), as defined below.
2. If *target* is the top of the document, then [scroll to the beginning of the document](#) for the [Document](#), and abort these steps. [\[CSSOM-VIEW\]](#)
3. Use the [scroll an element into view](#) algorithm to scroll *target* into view, with the *align to top flag* set. [\[CSSOM-VIEW\]](#)
4. Run the [focusing steps](#) for that element, with the [Document's viewport](#) as the *fallback target*.
5. Move the [sequential focus navigation starting point](#) to *target*.

The indicated part of the document is the one that the fragment identifier, if any, identifies. The semantics of the fragment identifier in terms of mapping it to a specific DOM Node is defined by the specification that defines the [MIME type](#) used by the [Document](#) (for example, the processing of fragment identifiers for [XML MIME types](#) is the responsibility of RFC7303). [\[RFC7303\]](#)

For HTML documents (and [HTML MIME types](#)), the following processing model must be followed to determine what [the indicated part of the document](#) is.

1. Apply the [URL parser](#) algorithm to the [URL](#), and let *fragid* be the [fragment](#) component of the [resulting URL record](#).
2. If *fragid* is the empty string, then [the indicated part of the document](#) is the top of the document; stop the algorithm here.
3. Let *fragid bytes* be the result of percent decoding *fragid*.
4. Let *decoded fragid* be the result of running [UTF-8 decode without BOM or fail](#) on *fragid bytes*. If *decoded fragid* is failure, jump to the step labeled no *decoded fragid*.
5. If there is an element in the DOM that has an [ID](#) exactly equal to *decoded fragid*, then the first such element in [tree order](#) is [the indicated part of the document](#); stop the algorithm here.
6. *No decoded fragid*: If there is an [<a>](#) element in the DOM that has a name attribute whose value is exactly equal to *fragid* (not *decoded fragid*), then the first such element in [tree order](#) is [the indicated part of the document](#); stop the algorithm here.
7. If *fragid* is an [ASCII case-insensitive](#) match for the string top, then [the indicated part of the document](#) is the top of the document; stop the algorithm here.
8. Otherwise, there is no [indicated part of the document](#).

For the purposes of the interaction of HTML with Selectors' `:target` pseudo-class, the *target element* is [the indicated part of the document](#), if that is an element; otherwise there is no *target element*. [\[CSS3-SELECTORS\]](#)

The [task source](#) for the task mentioned in this section must be the [DOM manipulation task source](#).

§ 6.7.10. History traversal

When a user agent is required to **traverse the history** to a *specified entry*, optionally with [replacement enabled](#), and optionally with the *non-blocking events* flag set, the user agent must act as follows.

NOTE:

This algorithm is not just invoked when [explicitly going back or forwards in the session history](#) — it is also invoked in other situations, for example when [navigating a browsing context](#), as part of [updating the session history with the new page](#).

1. If there is no longer a [Document](#) object for the entry in question, [navigate](#) the [browsing context](#) to the resource for that entry to perform an [entry update](#) of that entry, and abort these steps. The "navigate" algorithm reinvokes this "traverse" algorithm to complete the traversal, at which point there is a [Document](#) object and so this step gets skipped. The navigation must be done using the same [source browsing context](#) as was used the first time this entry was created. (This can never happen with [replacement enabled](#).)

NOTE:

If the resource was obtained using a non-idempotent action, for example a POST [form submission](#), or if the resource is no longer available, for example because the computer is now offline and the page wasn't cached, navigating to it again might not be possible. In this case, the navigation will result in a different page than previously; for example, it might be an error message explaining the problem or offering to resubmit the form.

2. If the [current entry](#)'s title was not set by the `pushState()` or `replaceState()` methods, then set its title to the value returned by the `document.title` IDL attribute.
3. If appropriate, update the [current entry](#) in the [browsing context](#)'s [Document](#) object's History object to reflect any state that the user agent wishes to persist. The entry is then said to be [an entry with persisted user state](#).
4. If the *specified entry* has a different [Document](#) object than the [current entry](#), then run the following substeps:
 1. Remove any [tasks](#) queued by the [history traversal task source](#) that are associated with any [Document](#) objects in the [top-level browsing context](#)'s [document family](#).
 2. If the [origin](#) of the [Document](#) of the *specified entry* is not the same as the [origin](#) of the [Document](#) of the [current entry](#), then run the following sub-sub-steps:
 1. The current [browsing context name](#) must be stored with all the entries in the history that are associated with [Document](#) objects with the [same origin](#) as the [active document](#) and that are contiguous with the [current entry](#).
 2. If the browsing context is a [top-level browsing context](#), but not an [auxiliary browsing context](#), then the browsing context's [browsing context name](#) must be unset.
 3. Make the *specified entry*'s [Document](#) object the [active document](#) of the [browsing context](#).
 4. If the *specified entry* has a [browsing context name](#) stored with it, then run the following sub-sub-steps:
 1. Set the browsing context's [browsing context name](#) to the name stored with the specified entry.
 2. Clear any [browsing context names](#) stored with all entries in the history that are associated with [Document](#) objects with the [same origin](#) as the new [active document](#) and that are contiguous with the specified entry.
5. If the *specified entry*'s [Document](#) has any form controls whose [autofill field name](#) is "off", invoke the [reset algorithm](#) of each of those elements.
6. If the [current document readiness](#) of the *specified entry*'s [Document](#) is "complete", [queue a task](#) to run the following sub-sub-steps:

1. If the [Document](#)'s [page showing](#) flag is true, then abort this task (i.e., don't fire the event below).
2. Set the [Document](#)'s [page showing](#) flag to true.
3. Run any **session history document visibility change steps** for [Document](#) that are defined by [other applicable specifications](#).

NOTE:

This is specifically intended for use by the Page Visibility specification. [\[PAGE-VISIBILITY\]](#)

4. [Fire](#) a [trusted](#) event with the name `pageshow` at the `Window` object of that [Document](#), with *target override* set to the [Document](#) object, using the `PageTransitionEvent` interface, with the `persisted` attribute initialized to true. This event must not bubble, must not be cancelable, and has no default action.
5. Set [the document's address](#) to the URL of the *specified entry*.
6. If the *specified entry* has a URL whose fragment identifier differs from that of the [current entry](#)'s when compared in a [case-sensitive](#) manner, and the two share the same [Document](#) object, then let *hash changed* be true, and let *old URL* be the URL of the [current entry](#) and *new URL* be the URL of the *specified entry*. Otherwise, let *hash changed* be false.
7. If the traversal was initiated with **replacement enabled**, remove the entry immediately before the *specified entry* in the session history.
8. If the *specified entry* is not [an entry with persisted user state](#), but its URL has a fragment identifier, [scroll to the fragment identifier](#).
9. If the entry is [an entry with persisted user state](#), the user agent may [restore persisted user state](#) and update aspects of the document and its rendering.
10. Let *targetRealm* be the [current Realm Record](#).
11. If the entry is a [state object](#) entry, let *state* be [StructuredClone](#)(that state object, *targetRealm*) of that state object. Otherwise, let *state* be null.
12. Set `history.state` to *state*.
13. Let *state changed* be true if the [Document](#) of the *specified entry* has a [latest entry](#), and that entry is not the *specified entry*; otherwise let it be false.
14. Let the [latest entry](#) of the [Document](#) of the *specified entry* be the *specified entry*.
15. If the *non-blocking events* flag is not set, then run the following steps [immediately](#). Otherwise, the *non-blocking events* flag is set; [queue a task](#) to run the following substeps instead.
 1. If *state changed* is true, [fire](#) a [trusted](#) event with the name `popstate` at the `Window` object of the [Document](#), using the `PopStateEvent` interface, with the `state` attribute initialized to the value of *state*. This event must bubble but not be cancelable and has no default action.
 2. If *hash changed* is true, then [fire](#) a [trusted](#) event with the name `hashchange` at the [browsing context](#)'s `Window` object, using the `HashChangeEvent` interface, with the `oldURL` attribute initialized to *old URL* and the `newURL` attribute initialized to *new URL*. This event must bubble but not be cancelable and has no default action.
16. The [current entry](#) is now the *specified entry*.

The [task source](#) for the tasks mentioned above is the [DOM manipulation task source](#).

§ 6.7.10.1. Persisted user state restoration

When the user agent is to **restore persisted user state** from a history entry, it must run the following steps immediately:

1. If the entry has a [scroll restoration mode](#), let *scrollRestoration* be that. Otherwise let *scrollRestoration* be "auto".
2. If *scrollRestoration* is "manual" the user agent should not restore the scroll position for the document, otherwise, it

may do so.

- Optionally, update other aspects of the document and its rendering, for instance values of form fields, that the user agent had previously recorded.

NOTE:

This can even include updating the `dir` attribute of `<textarea>` elements or `<input>` elements whose `type` attribute is in either the [Text](#) state or the [Search](#) state, if the persisted state includes the directionality of user input in such controls.

6.7.10.2. The *PopStateEvent* interface

```
[Constructor(DOMString type, optional PopStateEventInit eventInitDict), Exposed=(Window,Worker)]
interface PopStateEvent : Event {
  readonly attribute any state;
};

dictionary PopStateEventInit : EventInit {
  any state;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

***event* . state**

Returns a copy of the information that was provided to `pushState()` or `replaceState()`.

The **state** attribute must return the value it was initialized to. When the object is created, this attribute must be initialized to null. It represents the context information for the event, or null, if the state represented is the initial state of the [Document](#).

6.7.10.3. The *HashChangeEvent* interface

```
[Constructor(DOMString type, optional HashChangeEventInit eventInitDict), Exposed=
(Window,Worker)]
interface HashChangeEvent : Event {
  readonly attribute DOMString oldURL;
  readonly attribute DOMString newURL;
};

dictionary HashChangeEventInit : EventInit {
  DOMString oldURL;
  DOMString newURL;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

***event* . oldURL**

Returns the [URL](#) of the [session history entry](#) that was previously current.

***event* . newURL**

Returns the [URL](#) of the [session history entry](#) that is now current.

The **oldURL** attribute must return the value it was initialized to. When the object is created, this attribute must be initialized to null. It represents context information for the event, specifically the URL of the [session history entry](#) that was traversed from.

The **newURL** attribute must return the value it was initialized to. When the object is created, this attribute must be initialized to null. It represents context information for the event, specifically the URL of the [session history entry](#) that was traversed to.

§ 6.7.10.4. The *PageTransitionEvent* interface

```
[Constructor(DOMString type, optional PageTransitionEventInit eventInitDict), Exposed=
(Window,Worker)]
interface PageTransitionEvent : Event {
    readonly attribute boolean persisted;
};

dictionary PageTransitionEventInit : EventInit {
    boolean persisted;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

***event*.persisted**

For the *pageshow* event, returns false if the page is newly being loaded (and the *load* event will fire). Otherwise, returns true.

For the *pagehide* event, returns false if the page is going away for the last time. Otherwise, returns true, meaning that (if nothing conspires to make the page unsalvageable) the page might be reused if the user navigates back to this page.

Things that can cause the page to be unsalvageable include:

- `document.open()`
- Listening for *beforeunload* events
- Listening for *unload* events
- Having *iframes* that are not salvageable
- Active *WebSocket* objects
- [Aborting a Document](#)

The **persisted** attribute must return the value it was initialized to. When the object is created, this attribute must be initialized to false. It represents the context information for the event.

§ 6.7.11. Unloading documents

A [Document](#) has a *salvageable* state, which must initially be true, a **fired unload** flag, which must initially be false, and a **page showing** flag, which must initially be false. The *page showing* flag is used to ensure that scripts receive *pageshow* and *pagehide* events in a consistent manner (e.g., that they never receive two *pagehide* events in a row without an intervening *pageshow*, or vice versa).

[Event loops](#) have a **termination nesting level** counter, which must initially be zero.

When a user agent is to **prompt to unload a document**, it must run the following steps.

1. Increase the [event loop](#)'s *termination nesting level* by one.
2. Increase the [Document](#)'s *ignore-opens-during-unload counter* by one.
3. Let *event* be a new [trusted](#) *BeforeUnloadEvent* event object with the name *beforeunload*, which does not bubble but is cancelable.
4. *Dispatch*: [Dispatch event](#) at the [Document](#)'s *Window* object.
5. Decrease the [event loop](#)'s *termination nesting level* by one.
6. If any event listeners were triggered by the earlier *dispatch* step, then set the [Document](#)'s *salvageable* state to false.
7. If the [Document](#)'s *active sandboxing flag set* does not have its *sandboxed modals flag* set, and the *returnValue* attribute of the *event* object is not the empty string, or if the event was canceled, then the user agent should ask the user to confirm that they wish to unload the document.

The prompt shown by the user agent may include the string of the *returnValue* attribute, or some leading subset thereof. (A user agent may want to truncate the string to 1024 characters for display, for instance.)

The user agent must [pause](#) while waiting for the user's response.

If the user did not confirm the page navigation, then the user agent **refused to allow the document to be unloaded**.

8. If this algorithm was invoked by another instance of the "prompt to unload a document" algorithm (i.e., through the steps below that invoke this algorithm for all descendant browsing contexts), then jump to the step labeled *end*.
9. Let *descendants* be the [list of the descendant browsing contexts](#) of the [Document](#).
10. If *descendants* is not an empty list, then for each [browsing context](#) *b* in *descendants* run the following substeps:
 1. **Prompt to unload** the [active document](#) of the [browsing context](#) *b*. If the user [refused to allow the document to be unloaded](#), then the user implicitly also [refused to allow this document to be unloaded](#); jump to the step labeled *end*.
 2. If the *salvageable* state of the [active document](#) of the [browsing context](#) *b* is false, then set the *salvageable* state of *this* document to false also.
11. *End*: Decrease the [Document's ignore-opens-during-unload counter](#) by one.

When a user agent is to **unload a document**, it must run the following steps. These steps are passed an argument, *recycle*, which is either true or false, indicating whether the [Document](#) object is going to be re-used. (This is set by the `document.open()` method.)

1. Increase the [event loop's termination nesting level](#) by one.
2. Increase the [Document's ignore-opens-during-unload counter](#) by one.
3. If the [Document's page showing](#) flag is false, then jump to the step labeled *unload event* below (i.e., skip firing the pagehide event and don't rerun the [unloading document visibility change steps](#)).
4. Set the [Document's page showing](#) flag to false.
5. **Fire** a [trusted](#) event with the name `pagehide` at the Window object of the [Document](#), with *target override* set to the [Document](#) object, using the `PageTransitionEvent` interface, with the *persisted* attribute initialized to true if the [Document](#) object's *salvageable* state is true, and false otherwise. This event must not bubble, must not be cancelable, and has no default action.
6. Run any **unloading document visibility change steps** for [Document](#) that are defined by [other applicable specifications](#).

NOTE:

This is specifically intended for use by the Page Visibility specification. [\[PAGE-VISIBILITY\]](#)

7. *Unload event*: If the [Document's fired unload](#) flag is false, **fire a simple event** named `unload` at the [Document's](#) window object, with *target override* set to the [Document](#) object.
8. Decrease the [event loop's termination nesting level](#) by one.
9. If any event listeners were triggered by the earlier *unload event* step, then set the [Document](#) object's *salvageable* state to false and set the [Document's fired unload](#) flag to true.
10. Run any **unloading document cleanup steps** for [Document](#) that are defined by this specification and [other applicable specifications](#).
11. If this algorithm was invoked by another instance of the "unload a document" algorithm (i.e., by the steps below that invoke this algorithm for all descendant browsing contexts), then jump to the step labeled *end*.
12. Let *descendants* be the [list of the descendant browsing contexts](#) of the [Document](#).
13. If *descendants* is not an empty list, then for each [browsing context](#) *b* in *descendants* run the following substeps:
 1. **Unload** the [active document](#) of the [browsing context](#) *b* with the *recycle* parameter set to false.
 2. If the *salvageable* state of the [active document](#) of the [browsing context](#) *b* is false, then set the *salvageable* state of

this document to false also.

14. If both the [Document](#)'s *salvageable* state and `recycle` are false, then the [Document](#)'s *browsing context* must [discard the Document](#).
15. *End*: Decrease the [Document](#)'s *ignore-opens-during-unload counter* by one.

This specification defines the following **unloading document cleanup steps**. Other specifications can define more.

1. Make disappear any WebSocket objects that were created by the `WebSocket()` constructor from the [Document](#)'s Window object.

If this affected any WebSocket objects, then set [Document](#)'s *salvageable* state to false.
2. If the [Document](#)'s *salvageable* state is false, forcibly close any EventSource objects that whose constructor was invoked from the [Document](#)'s Window object.
3. If the [Document](#)'s *salvageable* state is false, empty the [Document](#)'s Window's *list of active timers*.

§ 6.7.11.1. The BeforeUnloadEvent interface

```
interface BeforeUnloadEvent : Event {
  attribute DOMString returnValue;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

`event.returnValue` [= *value*]

Returns the current return value of the event (the message to show the user).

Can be set, to update the message.

NOTE:

There are no BeforeUnloadEvent-specific initialization methods.

The **returnValue** attribute represents the message to show the user. When the event is created, the attribute must be set to the empty string. On getting, it must return the last value it was set to. On setting, the attribute must be set to the new value.

§ 6.7.12. Aborting a document load

If a [Document](#) is **aborted**, the user agent must run the following steps:

1. [Abort](#) the *active documents* of every *child browsing context*. If this results in any of those [Document](#) objects having their *salvageable* state set to false, then set this [Document](#)'s *salvageable* state to false also.
2. Cancel any instances of the [fetch](#) algorithm in the context of this [Document](#), discarding any *tasks queued* for them, and discarding any further data received from the network for them. If this resulted in any instances of the [fetch](#) algorithm being canceled or any *queued tasks* or any network data getting discarded, then set the [Document](#)'s *salvageable* state to false.
3. If the [Document](#) has an *active parser*, then [abort that parser](#) and set the [Document](#)'s *salvageable* state to false.

User agents may allow users to explicitly invoke the [abort a document](#) algorithm for a [Document](#). If the user does so, then, if that [Document](#) is an *active document*, the user agent should [queue a task](#) to [fire a simple event](#) named `abort` at that [Document](#)'s Window object before invoking the [abort](#) algorithm.

§ 6.7.13. Browser state

```
[NoInterfaceObject, Exposed=(Window, Worker)]
interface NavigatorOnLine {
  readonly attribute boolean onLine;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

`window.navigator.onLine`

Returns false if the user agent is definitely offline (disconnected from the network). Returns true if the user agent might be online.

The events `online` and `offline` are fired when the value of this attribute changes.

The `navigator.onLine` attribute must return false if the user agent will not contact the network when the user follows links or when a script requests a remote page (or knows that such an attempt would fail), and must return true otherwise.

When the value that would be returned by the `navigator.onLine` attribute of a `Window` or `WorkerGlobalScope` changes from true to false, the user agent must [queue a task](#) to [fire a simple event](#) named `offline` at the `Window` or `WorkerGlobalScope` object.

On the other hand, when the value that would be returned by the `navigator.onLine` attribute of a `Window` or `WorkerGlobalScope` changes from false to true, the user agent must [queue a task](#) to [fire a simple event](#) named `online` at the `Window` or `WorkerGlobalScope` object.

The [task source](#) for these [tasks](#) is the [networking task source](#).

NOTE:

This attribute is inherently unreliable. A computer can be connected to a network without having Internet access.

EXAMPLE 618

In this example, an indicator is updated as the browser goes online and offline.

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Online status</title>
    <script>
      function updateIndicator() {
        document.getElementById('indicator').textContent = navigator.onLine ? 'online' :
        'offline';
      }
    </script>
  </head>
  <body onload="updateIndicator()" ononline="updateIndicator()" onoffline="updateIndicator()">
    <p>The network is: <span>(state unknown)</span></p>
  </body>
</html>
```

§ 7. Web application APIs

§ 7.1. Scripting

§ 7.1.1. Introduction

Various mechanisms can cause author-provided executable code to run in the context of a document. These mechanisms include, but are probably not limited to:

- Processing of [script](#) elements.
- Navigating to [javascript: URLs](#).

- Event handlers, whether registered through the DOM using `addEventListener()`, by explicit [event handler content attributes](#), by [event handler IDL attributes](#), or otherwise.
- Processing of technologies like SVG that have their own scripting features.

§ 7.1.2. Enabling and disabling scripting

Scripting is enabled in a [browsing context](#) when all of the following conditions are true:

- The user agent supports scripting.
- The user has not disabled scripting for this [browsing context](#) at this time. (User agents may provide users with the option to disable scripting globally, or in a finer-grained manner, e.g. on a per-origin basis.)
- The [browsing context](#)'s [active document](#)'s [active sandboxing flag set](#) does not have its [sandboxed scripts browsing context flag](#) set.

Scripting is disabled in a [browsing context](#) when any of the above conditions are false (i.e., when scripting is not enabled).



Scripting is enabled for a *node* if the node's [node document](#) has a [browsing context](#), and [scripting is enabled](#) in that [browsing context](#).

Scripting is disabled for a *node* if there is no such [browsing context](#), or if [scripting is disabled](#) in that [browsing context](#).

§ 7.1.3. Processing model

§ 7.1.3.1. Definitions

A **script** is one of two possible structures. All scripts have:

A *settings object*

An [environment settings object](#), containing various settings that are shared with other [scripts](#) in the same context.

A **classic script** additionally has:

A *Source text*

A string containing a block of executable code to be evaluated as a JavaScript [Script](#).

Optionally, a *muted errors flag*

A flag which, if set, means that error information will not be provided for errors in this script (used to mute errors for cross-origin scripts, since that can leak private information).



An **environment settings object** specifies algorithms for obtaining the following:

A *realm execution context*

A [JavaScript execution context](#) shared by all [script](#) elements that use this settings object, i.e. all scripts in a given [JavaScript realm](#). When we [run a classic script](#) this execution context becomes the top of the [JavaScript execution context stack](#), on top of which another execution context specific to the script in question is pushed. (This setup ensures [ParseScript](#) knows which Realm to use.)

A *responsible browsing context*

A [browsing context](#) that is assigned responsibility for actions taken by the scripts that use this [environment settings object](#).

EXAMPLE 619

When a script creates and [navigates](#) a new [top-level browsing context](#), the [opener](#) attribute of the new [browsing context](#)'s [Window](#) object will be set to the [responsible browsing context](#)'s [WindowProxy](#) object.

A responsible event loop

An [event loop](#) that is used when it would not be immediately clear what event loop to use.

A responsible document

A [Document](#) that is assigned responsibility for actions taken by the scripts that use this [environment settings object](#).

EXAMPLE 620

For example, the [address](#) of the [responsible document](#) is used to set the [address](#) of the [Document](#) after it has been reset using [open\(\)](#).

If the [responsible event loop](#) is not a [browsing context event loop](#), then the [environment settings object](#) has no [responsible document](#).

An API URL character encoding

A character encoding used to encode URLs by APIs called by scripts that use this [environment settings object](#).

An API base URL

An [URL](#) used by APIs called by scripts that use this [environment settings object](#) to [parse URLs](#).

An origin

An instrument used in security checks.

A creation URL

An [absolute URL](#) representing the location of the resource with which the [environment settings object](#) is associated.

Note that this URL might be distinct from the [responsible document](#)'s [address](#), due to mechanisms such as [history.pushState\(\)](#).

An HTTPS state

An [HTTPS state value](#) representing the security properties of the network channel used to deliver the resource with which the [environment settings object](#) is associated.

An [environment settings object](#) also has an **outstanding rejected promises weak set** and an **about-to-be-notified rejected promises list**, used to track [unhandled promise rejections](#). The [outstanding rejected promises weak set](#) must not create strong references to any of its members, and implementations are free to limit its size, e.g., by removing old entries from it when new ones are added.

 7.1.3.2. Fetching scripts

The various script-fetching algorithms below have two hooks that may be customized by their callers:

- **Set up the request**, which takes a [request](#) which it may modify before the algorithm continues
- **Process the response**, which takes a [response](#) and must either return `true` or `false` to indicate success or failure, respectively

NOTE:

Service Workers is an example of a specification that runs these algorithms with its own options for the hooks.

[\[SERVICE-WORKERS\]](#)

To **fetch a classic script** for a `<script>` element *element*, given a *url*, a *CORS setting*, a *cryptographic nonce*, a *parser state*, a *settings object*, and a *character encoding*, run these steps. The algorithm will asynchronously complete with either null (on failure) or a new [classic script](#) (on success).

1. Let *request* be the result of [creating a potential-CORS request](#) given *url* and *CORS setting*.
2. Set *request*'s [client](#) to *settings object*, its [type](#) to "script", its [destination](#) to "script", its [cryptographic nonce metadata](#) to *cryptographic nonce*, and its [parser metadata](#) to *parser state*.

3. If the caller specified custom steps to [set up the request](#), perform them on *request*.
4. [Fetch](#) *request*.
5. Return from this algorithm, and run the remaining steps as part of the fetch's [process response](#) for the [response](#) *response*.

NOTE:

response can be either [CORS-same-origin](#) or [CORS-cross-origin](#). This only affects how error reporting happens.

6. If *response*'s [type](#) is "error", or *response*'s [status](#) is not an [ok status](#), asynchronously complete this algorithm with null, and abort these steps.
7. If the caller specified custom steps to [process the response](#), perform them on *response*. If they return false, complete this algorithm with null, and abort these steps.
8. If *response*'s [Content-Type metadata](#), if any, specifies a character encoding, and the user agent supports that encoding, then set *character encoding* to that encoding (ignoring the passed-in value).
9. Let *source text* be the result of [decoding](#) *response*'s [body](#) to Unicode, using *character encoding* as the fallback encoding.

NOTE:

The [decode](#) algorithm overrides *character encoding* if the file contains a BOM.

10. Let *script* be the result of [creating a classic script](#) using *source text* and *settings object*.
If *response* was [CORS-cross-origin](#), then pass the *muted errors* flag to the [create a classic script](#) algorithm as well.
11. Asynchronously complete this algorithm with *script*.

To **fetch a classic worker script** given a *url*, a *referrer*, a *settings object*, and a *destination*, run these steps. The algorithm will asynchronously complete with either null (on failure) or a new [classic script](#) (on success).

1. Let *request* be a new [request](#) whose [url](#) is *url*, [client](#) is *settings object*, [type](#) is "script", [destination](#) is *destination*, [referrer](#) is *referrer*, [mode](#) is "same-origin", [credentials mode](#) is "same-origin", [parser metadata](#) is "not parser-inserted", and whose [use-URL-credentials flag](#) is set.
2. If the caller specified custom steps to [set up the request](#), perform them on *request*.
3. [Fetch](#) *request*.
4. Return from this algorithm, and run the remaining steps as part of the fetch's [process response](#) for the [response](#) *response*.
5. If *response*'s [type](#) is "error", or *response*'s [status](#) is not an [ok status](#), asynchronously complete this algorithm with null, and abort these steps.
6. If the caller specified custom steps to [process the response](#), perform them on *response*. If they return false, complete this algorithm with null, and abort these steps.
7. Let *source text* be the result of [UTF-8 decoding](#) *response*'s [body](#).
8. Let *script* be the result of [creating a classic script](#) using *source text* and *settings object*.
9. Asynchronously complete this algorithm with *script*.

§ 7.1.3.3. Creating scripts

To **create a classic script**, given some script source, an [environment settings object](#), and an optional *muted errors* flag:

1. Let *script* be a new [classic script](#) that this algorithm will subsequently initialize.
2. Set *script*'s [settings object](#) to the [environment settings object](#) provided.

3. If [scripting is disabled](#) for the given [environment settings object](#)'s [responsible browsing context](#), set *script*'s [source text](#) to the empty string. Otherwise, set *script*'s [source text](#) to the supplied script source.
4. If the [muted errors](#) flag was set, then set *script*'s [muted errors](#) flag.
5. Return *script*.

7.1.3.4. Calling scripts

To **run a classic script** given a [classic script](#) *s* and an optional *rethrow errors* flag:

1. Let *settings* be the [settings object](#) of *s*.
2. [Check if we can run script](#) with *settings*. If this returns "do not run", then return undefined and abort these steps.
3. Let *realm* be *settings*'s [Realm](#).
4. [Prepare to run script](#) with *settings*.
5. Let *result* be [ParseScript](#)(*s*'s [source text](#), *realm*, *s*).
6. If *result* is a [List](#) of errors, set *result* to the first element of *result* and go to the step labeled *error*.
7. Let *evaluationStatus* be [ScriptEvaluation](#)(*result*).
8. If *evaluationStatus* is an abrupt completion, set *result* to *evaluationStatus*.[[value]] and go to the next step (labeled *Error*). If *evaluationStatus* is a normal completion, or if [ScriptEvaluation](#) does not complete because the user agent has [aborted the running script](#), skip to the step labeled *Cleanup*.
9. *Error*: At this point *result* must be an exception. Perform the following steps:
 1. If the [rethrow errors](#) flag is set and *s*'s [muted errors](#) flag is not set, rethrow *result*.
 2. If the [rethrow errors](#) flag is set and *s*'s [muted errors](#) flag is set, throw a [NetworkError](#) exception.
 3. If the [rethrow errors](#) flag is not set, [report the exception](#) given by *result* for the script *s*.
10. *Cleanup*: [Clean up after running script](#) with *settings*.
11. If *evaluationStatus* exists and is a normal completion, return *evaluationStatus*.[[value]]. Otherwise, script execution was unsuccessful, either because an error occurred during parsing, or an exception occurred during evaluation, or because it was [aborted prematurely](#).

The steps to **check if we can run script** with an [environment settings object](#) *settings* are as follows. They return either "run" or "do not run".

1. If the [global object](#) specified by *settings* is a [Window](#) object whose [Document](#) object is not [fully active](#), then return "do not run" and abort these steps.
2. If [scripting is disabled](#) for the [responsible browsing context](#) specified by *settings*, then return "do not run" and abort these steps.
3. Return "run".

The steps to **prepare to run script** with an [environment settings object](#) *settings* are as follows:

1. Increment *settings*'s [realm execution context](#)'s [entrance counter](#) by one.
2. Push *settings*'s [realm execution context](#) onto the [JavaScript execution context stack](#); it is now the [running JavaScript execution context](#).

The steps to **clean up after running script** with an [environment settings object](#) *settings* are as follows:

1. Assert: *settings*'s [realm execution context](#) is the [running JavaScript execution context](#).

2. Decrement `settings`'s [realm execution context](#)'s [entrance counter](#) by one.
3. Remove `settings`'s [realm execution context](#) from the [JavaScript execution context stack](#).
4. If the [JavaScript execution context stack](#) is now empty, [run the global script clean-up jobs](#). (These cannot run scripts.)
5. If the [JavaScript execution context stack](#) is now empty, [perform a microtask checkpoint](#). (If this runs scripts, these algorithms will be invoked reentrantly.)

NOTE:

These algorithms are not invoked by one script directly calling another, but they can be invoked reentrantly in an indirect manner, e.g., if a script dispatches an event which has event listeners registered.

The **running script** is the script in the `[[HostDefined]]` field in the Script component of the [running JavaScript execution context](#).

Each [unit of related similar-origin browsing contexts](#) has a **global script clean-up jobs list**, which must initially be empty. A global script clean-up job cannot run scripts, and cannot be sensitive to the order in which other clean-up jobs are executed. The File API uses this to release [blob:](#) URLs. [\[FILEAPI\]](#)

When the user agent is to **run the global script clean-up jobs**, the user agent must perform each of the jobs in the [global script clean-up jobs list](#) and then empty the list.

7.1.3.5. Realms, settings objects, and global objects

A **global object** is a JavaScript object that is the `[[globalObject]]` field of a [JavaScript realm](#).

NOTE:

In this specification, all [JavaScript realms](#) are [initialized](#) with [global objects](#) that are either [Window](#) or [WorkerGlobalScope](#) objects.

There is always a 1:1:1 mapping between [JavaScript realms](#), [global objects](#), and [environment settings objects](#):

- A [JavaScript realm](#) has a `[[HostDefined]]` field, which contains **the Realm's settings object**.
- A [JavaScript realm](#) has a `[[globalObject]]` field, which contains **the Realm's global object**.
- Each [global object](#) in this specification is created during the [initialization](#) of a corresponding [JavaScript realm](#), known as **the global object's Realm**.
- Each [global object](#) in this specification is created alongside a corresponding [environment settings object](#), known as its [relevant settings object](#).
- An [environment settings object](#)'s [realm execution context](#)'s Realm component is **the environment settings object's Realm**.
- An [environment settings object](#)'s [Realm](#) then has a `[[globalObject]]` field, which contains **the environment settings object's global object**.

When defining algorithm steps throughout this specification, it is often important to indicate what [JavaScript realm](#) is to be used—or, equivalently, what [global object](#) or [environment settings object](#) is to be used. In general, there are at least four possibilities:

Entry

This corresponds to the script that initiated the currently running script action: i.e., the function or script that the user agent called into when it called into author code.

Incumbent

This corresponds to the most-recently-entered author function or script on the stack.

Current

This corresponds to the currently-running function object, including built-in user-agent functions which might not be

implemented as JavaScript. (It is derived from the current [JavaScript realm](#).)

Relevant

Every [platform object](#) has a [relevant Realm](#). When writing algorithms, the most prominent [platform object](#) whose [relevant Realm](#) might be important is the `this` value of the currently-running function object. In some cases, there can be other important [relevant Realms](#), such as those of any arguments.

Note how the [entry](#), [incumbent](#), and [current](#) concepts are usable without qualification, whereas the [relevant](#) concept must be applied to a particular [platform object](#).

EXAMPLE 621

Consider the following pages, with `a.html` being loaded in a browser window, `b.html` being loaded in an `<iframe>` as shown, and `c.html` and `d.html` omitted (they can simply be empty documents):

```
<!-- a/a.html -->
<!DOCTYPE HTML>
<html lang="en">
<title>Entry page</title>

<iframe src="b.html"></iframe>
<button onclick="frames[0].hello()">Hello</button>

<!-- b.html -->
<!DOCTYPE HTML>
<html lang="en">
<title>Incumbent page</title>

<iframe src="c.html" id="c"></iframe>
<iframe src="d.html" id="d"></iframe>

<script>
const c = document.querySelector("#c").contentWindow;
const d = document.querySelector("#d").contentWindow;

window.hello = () => {
  c.print.call(d);
};
</script>
```

Each page has its own [browsing context](#), and thus its own [JavaScript realm](#), [global object](#), and [environment settings object](#).

When the `print()` method is called in response to pressing the button in `a.html`, then:

- The [entry Realm](#) is that of `a.html`.
- The [incumbent Realm](#) is that of `b.html`.
- The [current Realm](#) is that of `c.html` (since it is the `print()` method from `c.html` whose code is running).
- The [relevant Realm](#) of the object on which the `print()` method is being called is that of `d.html`.

⚠Warning! The [incumbent](#) and [entry](#) concepts should not be used by new specifications, and we are considering whether we can remove almost all existing uses

NOTE:

Currently, the [incumbent](#) concept is used in some security checks, and the [entry](#) concept is sometimes used to obtain, amongst other things, the [API base URL](#) to [parse a URL](#), used in scripts running in that [unit of related similar-origin browsing contexts](#).

In general, the [current](#) concept is what should be used by specifications going forward. There is an important exception, however. If an algorithm is creating an object that is to be persisted and returned multiple times (instead of simply returned to author code right away, and never vended again), it should use the [relevant](#) concept with regard to the object on which the method in question is being executed. This prevents cross-realm calls from causing an object to store objects created in the "wrong" realm.

EXAMPLE 622

The `navigator.getBattery()` method creates promises in the [relevant Realm](#) for the [Navigator](#) object on which it is invoked. This has the following impact: [\[BATTERY-STATUS\]](#)

```
<!-- outer.html -->
<!DOCTYPE html>
<html lang="en">
<title>Relevant Realm demo: outer page</title>
<script>
  function doTest() {
    const promise = navigator.getBattery.call(frames[0].navigator);

    console.log(promise instanceof Promise);           // logs false
    console.log(promise instanceof frames[0].Promise); // logs true

    frames[0].hello();
  }
</script>
<iframe src="inner.html" onload="doTest()"></iframe>

<!-- inner.html -->
<!DOCTYPE html>
<html lang="en">
<title>Relevant Realm demo: inner page</title>
<script>
  function hello() {
    const promise = navigator.getBattery();

    console.log(promise instanceof Promise);           // logs true
    console.log(promise instanceof parent.Promise);    // logs false
  }
</script>
```

If the algorithm for the `getBattery()` method had instead used the [current Realm](#), all the results would be reversed. That is, after the first call to `getBattery()` in `outer.html`, the [Navigator](#) object in `inner.html` would be permanently storing a [Promise](#) object created in `outer.html`'s [JavaScript realm](#), and calls like that inside the `hello()` function would thus return a promise from the "wrong" realm. Since this is undesirable, the algorithm instead uses the [relevant Realm](#), giving the sensible results indicated in the comments above.

The rest of this section deals with formally defining the [entry](#), [incumbent](#), [current](#), and [relevant](#) concepts.

§ 7.1.3.5.1. ENTRY

All [realm execution contexts](#) must contain, as part of their code evaluation state, an **entrance counter** value, which is initially zero. In the process of [calling scripts](#), this value will be incremented and decremented.

With this in hand, we define the **entry execution context** to be the most recently pushed entry in the [JavaScript execution context stack](#) whose [entrance counter](#) value is greater than zero. The **entry Realm** is the [entry execution context](#)'s Realm component.

Then, the **entry settings object** is the [environment settings object](#) of the [entry Realm](#).

Similarly, the **entry global object** is the [global object](#) of the [entry Realm](#).

§ 7.1.3.5.2. INCUMBENT

The **incumbent settings object** is determined as follows:

1. Let *script* be the result of JavaScript's [GetActiveScriptOrModule\(\)](#) abstract operation.
2. If *script* is null, abort these steps; there is no [incumbent settings object](#).
3. Return the [settings object](#) of the `<script>` in *script*'s `[[HostDefined]]` field.

Then, the **incumbent Realm** is the [Realm](#) of the [incumbent settings object](#).

Similarly, the **incumbent global object** is the [global object](#) of the [incumbent settings object](#).

§ 7.1.3.5.3. CURRENT

The JavaScript specification defines the [current Realm Record](#), sometimes abbreviated to the "current Realm". [\[ECMA-262\]](#)

Then, the **current settings object** is the [environment settings object](#) of the [current Realm Record](#).

Similarly, the **current global object** is the [global object](#) of the [current Realm Record](#).

§ 7.1.3.5.4. RELEVANT

The **relevant settings object** for a [platform object](#) is defined as follows:

↪ If the object is a [global object](#)

Each [global object](#) in this specification is created alongside a corresponding [environment settings object](#); that is its [relevant settings object](#).

↪ Otherwise

The [relevant settings object](#) for a non-global [platform object](#) *o* is the [environment settings object](#) whose [global object](#) is the global object of the [global environment associated with](#) *o*.

NOTE:

The "[global environment associated with](#)" concept is from the olden days, before the modern JavaScript specification and its concept of [realms](#). We expect that as the Web IDL specification gets updated, every [platform object](#) will have a [Realm](#) associated with it, and this definition can be re-cast in those terms. [\[ECMA-262\]](#) [\[WEBIDL\]](#)

Then, the **relevant Realm** for a [platform object](#) is the [Realm](#) of its [relevant settings object](#).

Similarly, the **relevant global object** for a [platform object](#) is the [global object](#) of its [relevant settings object](#).

§ 7.1.3.6. Killing scripts

Although the JavaScript specification does not account for this possibility, it's sometimes necessary to **abort a running script**. This causes any [ScriptEvaluation](#) to cease immediately, emptying the [JavaScript execution context stack](#) without triggering any of the normal mechanisms like `finally` blocks. [\[ECMA-262\]](#)

User agents may impose resource limitations on scripts, for example CPU quotas, memory limits, total execution time limits, or bandwidth limitations. When a script exceeds a limit, the user agent may either throw a [QuotaExceededError](#) exception, [abort the script](#) without an exception, prompt the user, or throttle script execution.

EXAMPLE 623

For example, the following script never terminates. A user agent could, after waiting for a few seconds, prompt the user to either terminate the script or let it continue.

```
<script>
  while (true) { /* loop */ }
</script>
```

User agents are encouraged to allow users to disable scripting whenever the user is prompted either by a script (e.g., using the [window.alert\(\)](#) API) or because of a script's actions (e.g., because it has exceeded a time limit).

If scripting is disabled while a script is executing, the script should be terminated immediately.

User agents may allow users to specifically disable scripts just for the purposes of closing a [browsing context](#).

EXAMPLE 624

For example, the prompt mentioned in the example above could also offer the user with a mechanism to just close the page entirely, without running any unload event handlers.

§ 7.1.3.7. *Integration with the JavaScript job queue*

The JavaScript specification defines the JavaScript job and job queue abstractions in order to specify certain invariants about how promise operations execute with a clean [JavaScript execution context stack](#) and in a certain order. However, as of the time of this writing the definition of [EnqueueJob](#) in that specification are not sufficiently flexible to integrate with HTML as a host environment. [\[ECMA-262\]](#)

NOTE:

This is not strictly true. It is in fact possible, by taking liberal advantage of the many "implementation defined" sections of the algorithm, to contort it to our purposes. However, the end result is a mass of messy indirection and workarounds that essentially bypasses the job queue infrastructure entirely, albeit in a way that is technically sanctioned within the bounds of implementation-defined behavior. We do not take this path, and instead introduce the following [willful violation](#).

As such, user agents must instead use the following definition in place of that in the JavaScript specification. These ensure that the promise jobs enqueued by the JavaScript specification are properly integrated into the user agent's [event loops](#).

§ 7.1.3.7.1. **ENQUEUEJOB**(*QUEUEName*, *JOB*, *ARGUMENTS*)

When the JavaScript specification says to call the EnqueueJob abstract operation, the following algorithm must be used in place of JavaScript's [EnqueueJob](#):

1. Assert: *queueName* is "PromiseJobs". ("ScriptJobs" must not be used by user agents.)
2. Let *settings* be the [settings object](#) of *job*.[[Realm]]
3. [Queue a microtask](#), on *settings*'s [responsible event loop](#), to perform the following steps:
 1. [Check if we can run script](#) with *settings*. If this returns "do not run" then abort these steps.
 2. [Prepare to run script](#) with *settings*.
 3. Let *result* be the result of performing the abstract operation specified by *job*, using the elements of *arguments* as its arguments.
 4. [Clean up after running script](#) with *settings*.
 5. If *result* is an abrupt completion, [report the exception](#) given by *result*.[[value]].

§ 7.1.3.8. Runtime script errors

When the user agent is required to **report an error** for a particular [script](#) with a particular position `line:col`, using a particular target `target`, it must run these steps, after which the error is either **handled** or **not handled**:

1. If `target` is [in error reporting mode](#), then abort these steps; the error is [not handled](#).
2. Let `target` be **in error reporting mode**.
3. Let `message` be a user-agent-defined string describing the error in a helpful manner.
4. Let `error object` be the object that represents the error: in the case of an uncaught exception, that would be the object that was thrown; in the case of a JavaScript error that would be an [Error](#) object. If there is no corresponding object, then the null value must be used instead.
5. Let `location` be an [absolute URL](#) that corresponds to the resource from which `script` was obtained.

NOTE:

The resource containing the script will typically be the file from which the [Document](#) was parsed, e.g., for inline `<script>` elements or [event handler content attributes](#); or the JavaScript file that the script was in, for external scripts. Even for dynamically-generated scripts, user agents are strongly encouraged to attempt to keep track of the original source of a script. For example, if an external script uses the `document.write()` API to insert an inline `<script>` element during parsing, the URL of the resource containing the script would ideally be reported as being the external script, and the line number might ideally be reported as the line with the `document.write()` call or where the string passed to that call was first constructed. Naturally, implementing this can be somewhat non-trivial.

NOTE:

User agents are similarly encouraged to keep careful track of the original line numbers, even in the face of `document.write()` calls mutating the document as it is parsed, or [event handler content attributes](#) spanning multiple lines.

6. If `script` has [muted errors](#), then set `message` to "Script error.", set `location` to the empty string, set `line` and `col` to 0, and set `error object` to null.
7. Let `event` be a new [trusted ErrorEvent](#) object that does not bubble but is cancelable, and which has the event name `error`.
8. Initialize `event`'s [message](#) attribute to `message`.
9. Initialize `event`'s [filename](#) attribute to `location`.
10. Initialize `event`'s [lineno](#) attribute to `line`.
11. Initialize `event`'s [colno](#) attribute to `col`.
12. Initialize `event`'s [error](#) attribute to `error object`.
13. [Dispatch](#) `event` at `target`.
14. Let `target` no longer be [in error reporting mode](#).
15. If `event` was canceled, then the error is [handled](#). Otherwise, the error is [not handled](#).

NOTE:

Returning true cancels `event` per [the event handler processing algorithm](#).

§ 7.1.3.8.1. RUNTIME SCRIPT ERRORS IN DOCUMENTS

When the user agent is to **report an exception** *E*, the user agent must [report the error](#) for the relevant [script](#), with the problematic position (line number and column number) in the resource containing the script, using the [global object](#) specified by the script's [settings object](#) as the target. If the error is still [not handled](#) after this, then the error may be reported to a developer console.

§ 7.1.3.8.2. THE `ErrorEvent` INTERFACE

```
[Constructor(DOMString type, optional ErrorEventInit eventInitDict), Exposed=(Window, Worker)]
interface ErrorEvent : Event {
  readonly attribute DOMString message;
  readonly attribute DOMString filename;
  readonly attribute unsigned long lineno;
  readonly attribute unsigned long colno;
  readonly attribute any error;
};

dictionary ErrorEventInit : EventInit {
  DOMString message = "";
  DOMString filename = "";
  unsigned long lineno = 0;
  unsigned long colno = 0;
  any error = null;
};
```

The **message** attribute must return the value it was initialized to. It represents the error message.

The **filename** attribute must return the value it was initialized to. It represents the [absolute URL](#) of the script in which the error originally occurred.

The **lineno** attribute must return the value it was initialized to. It represents the line number where the error occurred in the script.

The **colno** attribute must return the value it was initialized to. It represents the column number where the error occurred in the script.

The **error** attribute must return the value it was initialized to. Where appropriate, it is set to the object representing the error (e.g., the exception object in the case of an uncaught DOM exception).

§ 7.1.3.9. Unhandled promise rejections

⚠Warning! There is only one known native implementation of the `unhandledrejection` and `rejectionhandled` events (Chrome/Blink). Therefore these features should not be relied upon.

In addition to synchronous [runtime script errors](#), scripts may experience asynchronous promise rejections, tracked via the `unhandledrejection` and `rejectionhandled` events.

When the user agent is to **notify about rejected promises** on a given [environment settings object](#) *settings object*, it must run these steps:

1. Let *list* be a copy of *settings object*'s [about-to-be-notified rejected promises list](#).
2. If *list* is empty, abort these steps.
3. Clear *settings object*'s [about-to-be-notified rejected promises list](#).
4. [Queue a task](#) to run the following substep:
 1. For each promise *p* in *list*:
 1. If *p*'s `[[PromiseIsHandled]]` internal slot is true, continue to the next iteration of the loop.
 2. Let *event* be a new [trusted](#) `PromiseRejectionEvent` object that does not bubble but is cancelable, and which has the event name `unhandledrejection`.
 3. Initialise *event*'s [promise](#) attribute to *p*.

4. Initialise `event`'s `reason` attribute to the value of `p`'s `[[PromiseResult]]` internal slot.
5. `Dispatch event` at `settings object`'s `global object`.
6. If the event was canceled, then the promise rejection is `handled`. Otherwise, the promise rejection is `not handled`.
7. If `p`'s `[[PromiseIsHandled]]` internal slot is false, add `p` to `settings object`'s `outstanding rejected promises weak set`.

This algorithm results in promise rejections being marked as **handled** or **not handled**. These concepts parallel `handled` and `not handled` script errors. If a rejection is still `not handled` after this, then the rejection may be reported to a developer console.

§ 7.1.3.9.1. THE `HostPromiseRejectionTracker` IMPLEMENTATION

ECMAScript contains an implementation-defined `HostPromiseRejectionTracker`(`promise`, `operation`) abstract operation. User agents must use the following implementation: [ECMA-262]

1. Let `script` be the `running script`.
2. If `script` has `muted errors`, terminate these steps.
3. Let `settings object` be `script`'s `settings object`.
4. If `operation` is "reject",
 1. Add `promise` to `settings object`'s `about-to-be-notified rejected promises list`.
5. If `operation` is "handle",
 1. If `settings object`'s `about-to-be-notified rejected promises list` contains `promise`, remove `promise` from that list and abort these steps.
 2. If `settings object`'s `outstanding rejected promises weak set` does not contain `promise`, abort these steps.
 3. Remove `promise` from `settings object`'s `outstanding rejected promises weak set`.
 4. `Queue a task` to run the following steps:
 1. Let `event` be a new `trusted PromiseRejectionEvent` object that does not bubble and is not cancelable, and which has the event name `rejectionhandled`.
 2. Initialise `event`'s `promise` attribute to `promise`.
 3. Initialise `event`'s `reason` attribute to the value of `promise`'s `[[PromiseResult]]` internal slot.
 4. `Dispatch event` at `settings object`'s `global object`.

§ 7.1.3.9.2. THE `PromiseRejectionEvent` INTERFACE

```
[Constructor(DOMString type, PromiseRejectionEventInit eventInitDict), Exposed=(Window,Worker)]
interface PromiseRejectionEvent : Event {
  readonly attribute Promise<any> promise;
  readonly attribute any reason;
};

dictionary PromiseRejectionEventInit : EventInit {
  required Promise<any> promise;
  any reason;
};
```

The **promise** attribute must return the value it was initialized to. It represents the promise which this notification is about.

The **reason** attribute must return the value it was initialized to. It represents the rejection reason for the promise.

§ 7.1.3.10. *HostEnsureCanCompileStrings*(*callerRealm*, *calleeRealm*)

JavaScript contains an implementation-defined [HostEnsureCanCompileStrings](#)(*callerRealm*, *calleeRealm*) abstract operation. User agents must use the following implementation: [ECMA-262]

1. Perform ? [EnsureCSPDoesNotBlockStringCompilation](#)(*callerRealm*, *calleeRealm*). [CSP3]

§ 7.1.4. Event loops

§ 7.1.4.1. Definitions

To coordinate events, user interaction, scripts, rendering, networking, and so forth, user agents must use **event loops** as described in this section. There are two kinds of event loops: those for [browsing contexts](#), and those for [workers](#).

There must be at least one [browsing context event loop](#) per user agent, and at most one per [unit of related similar-origin browsing contexts](#).

NOTE:

When there is more than one [event loop](#) for a [unit of related browsing contexts](#), complications arise when a [browsing context](#) in that group is [navigated](#) such that it switches from one [unit of related similar-origin browsing contexts](#) to another. This specification does not currently describe how to handle these complications.

A [browsing context event loop](#) always has at least one [browsing context](#). If such an [event loop](#)'s [browsing contexts](#) all go away, then the [event loop](#) goes away as well. A [browsing context](#) always has an [event loop](#) coordinating its activities.

[Worker event loops](#) are simpler: each worker has one [event loop](#), and the [worker processing model](#) manages the [event loop](#)'s lifetime.



An [event loop](#) has one or more **task queues**. A [task queue](#) is an ordered list of **tasks**, which are algorithms that are responsible for such work as:

Events

Dispatching an [Event](#) object at a particular [EventTarget](#) object is often done by a dedicated task.

NOTE:

Not all events are dispatched using the [task queue](#), many are dispatched during other tasks.

Parsing

The [HTML parser](#) tokenizing one or more bytes, and then processing any resulting tokens, is typically a task.

Callbacks

Calling a callback is often done by a dedicated task.

Using a resource

When an algorithm [fetches](#) a resource, if the fetching occurs in a non-blocking fashion then the processing of the resource once some or all of the resource is available is performed by a task.

Reacting to DOM manipulation

Some elements have tasks that trigger in response to DOM manipulation, e.g., when that element is [inserted into the document](#).

Each [task](#) in a [browsing context event loop](#) is associated with a [Document](#); if the task was queued in the context of an element, then it is the element's [node document](#); if the task was queued in the context of a [browsing context](#), then it is the [browsing context](#)'s [active document](#) at the time the task was queued; if the task was queued by or for a [script](#) then the docu-

ment is the [responsible document](#) specified by the script's [settings object](#).

A [task](#) is intended for a specific [event loop](#): the [event loop](#) that is handling [tasks](#) for the [task](#)'s associated [Document](#) or [Worker](#).

When a user agent is to **queue a task**, it must add the given task to one of the [task queues](#) of the relevant [event loop](#).

Each [task](#) is defined as coming from a specific [task source](#). All the tasks from one particular [task source](#) and destined to a particular [event loop](#) (e.g., the callbacks generated by timers of a [Document](#), the events fired for mouse movements over that [Document](#), the tasks queued for the parser of that [Document](#)) must always be added to the same [task queue](#), but [tasks](#) from different [task sources](#) may be placed in different [task queues](#).

EXAMPLE 625

For example, a user agent could have one [task queue](#) for mouse and key events (the [user interaction task source](#)), and another for everything else. The user agent could then give keyboard and mouse events preference over other tasks three quarters of the time, keeping the interface responsive but not starving other task queues, and never processing events from any one [task source](#) out of order.

Each [event loop](#) has a **currently running task**. Initially, this is null. It is used to handle reentrancy. Each [event loop](#) also has a **performing a microtask checkpoint flag**, which must initially be false. It is used to prevent reentrant invocation of the [perform a microtask checkpoint](#) algorithm.

7.1.4.2. Processing model

An [event loop](#) must continually run through the following steps for as long as it exists:

1. Select the oldest [task](#) on one of the [event loop](#)'s [task queues](#), if any, ignoring, in the case of a [browsing context event loop](#), tasks whose associated [Documents](#) are not [fully active](#). The user agent may pick any [task queue](#). If there is no task to select, then jump to the *Microtasks* step below.
2. Set the [event loop](#)'s [currently running task](#) to the [task](#) selected in the previous step.
3. *Run*: Run the selected [task](#).
4. Set the [event loop](#)'s [currently running task](#) back to null.
5. Remove the task that was run in the *Run* step above from its [task queue](#).
6. *Microtasks*: [Perform a microtask checkpoint](#).
7. *Update the rendering*: If this [event loop](#) is a [browsing context event loop](#) (as opposed to a [Worker event loop](#)), then run the following substeps.
 1. Let *now* be the value that would be returned by the [Performance](#) object's [now\(\)](#) method. [HR-TIME-2]
 2. Let *docs* be the list of [Document](#) objects associated with the [event loop](#) in question, sorted arbitrarily except that the following conditions must be met:
 - Any [Document](#) *B* that is [nested through](#) a [Document](#) *A* must be listed after *A* in the list.
 - If there are two documents *A* and *B* whose [browsing contexts](#) are both [nested browsing contexts](#) and their [browsing context containers](#) are both elements in the same [Document](#) *C*, then the order of *A* and *B* in the list must match the relative [tree order](#) of their respective [browsing context containers](#) in *C*.

In the steps below that iterate over *docs*, each [Document](#) must be processed in the order it is found in the list.

3. If there is a [top-level browsing context](#) *B* that the user agent believes would not benefit from having its rendering updated at this time, then remove from *docs* all [Document](#) objects whose [browsing context](#)'s [top-level browsing context](#) is *B*.

NOTE:

Whether a [top-level browsing context](#) would benefit from having its rendering updated depends on various factors, such as the update frequency. For example, if the browser is attempting to achieve a 60 Hz refresh rate, then these steps are only necessary every 60th of a second (about 16.7ms). If the browser finds that a [top-level browsing context](#) is not able to sustain this rate, it might drop to a more sustainable 30Hz for that set of [Documents](#), rather than occasionally dropping frames. (This specification does not mandate any particular model for when to update the rendering.) Similarly, if a [top-level browsing context](#) is in the background, the user agent might decide to drop that page to a much slower 4Hz, or even less.

NOTE:

Another example of why a browser might skip updating the rendering is to ensure certain [tasks](#) are executed immediately after each other, with only [microtask checkpoints](#) interleaved (and without, e.g., [animation frame callbacks](#) interleaved). For example, a user agent might wish to coalesce timer callbacks together, with no intermediate rendering updates.

4. If there are a [nested browsing contexts](#) *B* that the user agent believes would not benefit from having their rendering updated at this time, then remove from *docs* all [Document](#) objects whose [browsing context](#) is in *B*.

NOTE:

As with [top-level browsing contexts](#), a variety of factors can influence whether it is profitable for a browser to update the rendering of [nested browsing contexts](#). For example, a user agent might wish to spend less resources rendering third-party content, especially if it is not currently visible to the user or if resources are constrained. In such cases, the browser could decide to update the rendering for such content infrequently or never.

5. For each [fully active Document](#) in *docs*, [run the resize steps](#) for that [Document](#), passing in *now* as the timestamp. [\[CSSOM-VIEW\]](#)
 6. For each [fully active Document](#) in *docs*, [run the scroll steps](#) for that [Document](#), passing in *now* as the timestamp. [\[CSSOM-VIEW\]](#)
 7. For each [fully active Document](#) in *docs*, [evaluate media queries and report changes](#) for that [Document](#), passing in *now* as the timestamp. [\[CSSOM-VIEW\]](#)
 8. For each [fully active Document](#) in *docs*, [run CSS animations and send events](#) for that [Document](#), passing in *now* as the timestamp. [\[CSS3-ANIMATIONS\]](#)
 9. For each [fully active Document](#) in *docs*, [run the fullscreen rendering steps](#) for that [Document](#), passing in *now* as the timestamp. [\[FULLSCREEN\]](#)
 10. For each [fully active Document](#) in *docs*, [run the animation frame callbacks](#) for that [Document](#), passing in *now* as the timestamp.
 11. For each [fully active Document](#) in *docs*, update the rendering or user interface of that [Document](#) and its [browsing context](#) to reflect the current state.
8. If this is a [Worker event loop](#) (i.e., one running for a [WorkerGlobalScope](#)), but there are no [tasks](#) in the [event loop](#)'s [task queues](#) and the [WorkerGlobalScope](#) object's [closing](#) flag is true, then destroy the [event loop](#), aborting these steps, resuming the [run a worker](#) steps.
 9. Return to the first step of the [event loop](#).



Each [event loop](#) has a **microtask queue**. A **microtask** is a [task](#) that is originally to be queued on the [microtask queue](#) rather than a [task queue](#). There are two kinds of [microtasks](#): **solitary callback microtasks**, and **compound microtasks**.

NOTE:

This specification only has [solitary callback microtasks](#). Specifications that use [compound microtasks](#) have to take extra care to [wrap callbacks](#) to handle [spinning the event loop](#).

When an algorithm requires a [microtask](#) to be **queued**, it must be appended to the relevant [event loop](#)'s [microtask queue](#); the [task source](#) of such a [microtask](#) is the **microtask task source**.

NOTE:

It is possible for a [microtask](#) to be moved to a regular [task queue](#), if, during its initial execution, it [spins the event loop](#). In that case, the [microtask task source](#) is the [task source](#) used. Normally, the [task source](#) of a [microtask](#) is irrelevant.

When a user agent is to **perform a microtask checkpoint**, if the [performing a microtask checkpoint flag](#) is false, then the user agent must run the following steps:

1. Let the [performing a microtask checkpoint flag](#) be true.
2. *Microtask queue handling*: If the [event loop](#)'s [microtask queue](#) is empty, jump to the *Done* step below.
3. Select the oldest [microtask](#) on the [event loop](#)'s [microtask queue](#).
4. Set the [event loop](#)'s [currently running task](#) to the [task](#) selected in the previous step.
5. *Run*: Run the selected [task](#).

NOTE:

This might involve invoking scripted callbacks, which eventually calls the [clean up after running script](#) steps, which call this [perform a microtask checkpoint](#) algorithm again, which is why we use the [performing a microtask checkpoint flag](#) to avoid reentrancy.

6. Set the [event loop](#)'s [currently running task](#) back to null.
7. Remove the [microtask](#) run in the step above from the [microtask queue](#), and return to the *Microtask queue handling* step.
8. *Done*: For each [environment settings object](#) whose [responsible event loop](#) is this [event loop](#), [notify about rejected promises](#) on that [environment settings object](#).
9. Let the [performing a microtask checkpoint flag](#) be false.

If, while a [compound microtask](#) is running, the user agent is required to **execute a compound microtask subtask** to run a series of steps, the user agent must run the following steps:

1. Let *parent* be the [event loop](#)'s [currently running task](#) (the currently running [compound microtask](#)).
2. Let *subtask* be a new [task](#) that consists of running the given series of steps. The [task source](#) of such a [microtask](#) is the [microtask task source](#). This is a **compound microtask subtask**.
3. Set the [event loop](#)'s [currently running task](#) to *subtask*.
4. Run *subtask*.
5. Set the [event loop](#)'s [currently running task](#) back to *parent*.



When an algorithm running [in parallel](#) is to **await a stable state**, the user agent must [queue a microtask](#) that runs the following steps, and must then stop executing (execution of the algorithm resumes when the microtask is run, as described in the following steps):

1. Run the algorithm's **synchronous section**.
2. Resumes execution of the algorithm [in parallel](#), if appropriate, as described in the algorithm's steps.

NOTE:

Steps in [synchronous sections](#) are marked with ⌚.



When an algorithm says to **spin the event loop** until a condition *goal* is met, the user agent must run the following steps:

1. Let *task* be the [event loop](#)'s [currently running task](#).

NOTE:

This might be a [microtask](#), in which case it is a [solitary callback microtask](#). It could also be a [compound microtask subtask](#), or a regular [task](#) that is not a [microtask](#). It will not be a [compound microtask](#).

2. Let *task source* be *task*'s [task source](#).
3. Let *old stack* be a copy of the [JavaScript execution context stack](#).
4. Empty the [JavaScript execution context stack](#).
5. [Run the global script clean-up jobs](#).
6. [Perform a microtask checkpoint](#).
7. Stop *task*, allowing whatever algorithm that invoked it to resume, but continue these steps [in parallel](#).

NOTE:

This causes one of the following algorithms to continue: the [event loop](#)'s main set of steps, the [perform a microtask checkpoint](#) algorithm, or the [execute a compound microtask subtask](#) algorithm to continue.

8. Wait until the condition *goal* is met.
9. [Queue a task](#) to continue running these steps, using the [task source](#) *task source*. Wait until this new task runs before continuing these steps.
10. Replace the [JavaScript execution context stack](#) with the *old stack*.
11. Return to the caller.



Some of the algorithms in this specification, for historical reasons, require the user agent to **pause** while running a [task](#) until a condition *goal* is met. This means running the following steps:

1. If necessary, update the rendering or user interface of any [Document](#) or [browsing context](#) to reflect the current state.
2. Wait until the condition *goal* is met. While a user agent has a paused [task](#), the corresponding [event loop](#) must not run further [tasks](#), and any script in the currently running [task](#) must block. User agents should remain responsive to user input while paused, however, albeit in a reduced capacity since the [event loop](#) will not be doing anything.

§ 7.1.4.3. Generic task sources

The following [task sources](#) are used by a number of mostly unrelated features in this and other specifications.

The *DOM manipulation task source*

This [task source](#) is used for features that react to DOM manipulations, such as things that happen in a non-blocking fashion when an element is [inserted into the document](#).

The *user interaction task source*

This [task source](#) is used for features that react to user interaction, for example keyboard or mouse input.

Events sent in response to user input (e.g., click events) must be fired using [tasks queued](#) with the [user interaction task source](#). [\[UIEVENTS\]](#)

The *networking task source*

This [task source](#) is used for features that trigger in response to network activity.

The *history traversal task source*

This [task source](#) is used to queue calls to [history.back\(\)](#) and similar APIs.

§ 7.1.5. Events

§ 7.1.5.1. Event handlers

Many objects can have **event handlers** specified. These act as non-capture event listeners for the object on which they are specified. [\[DOM\]](#)

An [event handler](#) has a name, which always starts with "on" and is followed by the name of the event for which it is intended.

An [event handler](#) has a value, which is either null, or is a callback object, or is an [internal raw uncompiled handler](#). The [EventHandler](#) callback function type describes how this is exposed to scripts. Initially, an [event handler](#)'s value must be set to null.

Event handlers are exposed in one of two ways.

The first way, common to all event handlers, is as an [event handler IDL attribute](#).

The second way is as an [event handler content attribute](#). Event handlers on [html elements](#) and some of the event handlers on [Window](#) objects are exposed in this way.



An **event handler IDL attribute** is an IDL attribute for a specific [event handler](#). The name of the IDL attribute is the same as the name of the [event handler](#).

[Event handler IDL attributes](#), on setting, must set the corresponding [event handler](#) to their new value, and on getting, must return the result of [getting the current value of the event handler](#) in question (this can throw an exception, in which case the getting propagates it to the caller, it does not catch it).

If an [event handler IDL attribute](#) exposes an [event handler](#) of an object that doesn't exist, it must always return null on getting and must do nothing on setting.

NOTE:

This can happen in particular for [event handler IDL attribute](#) on `<body>` elements that do not have corresponding [Window](#) objects.

NOTE:

Certain event handler IDL attributes have additional requirements, in particular the `onmessage` attribute of [MessagePort](#) objects.



An **event handler content attribute** is a content attribute for a specific [event handler](#). The name of the content attribute is the same as the name of the [event handler](#).

[Event handler content attributes](#), when specified, must contain valid JavaScript code which, when parsed, would match the [FunctionBody](#) production after automatic semicolon insertion. [\[ECMA-262\]](#)

When an [event handler content attribute](#) is set, execute the following steps:

1. If the [Should element's inline behavior be blocked by Content Security Policy?](#) algorithm returns "Blocked" when executed upon the attribute's element "script attribute", and the attribute's value, then abort these steps. [\[CSP3\]](#)
2. Set the corresponding [event handler](#) to an [internal raw uncompiled handler](#) consisting of the attribute's new value and the script location where the attribute was set to this value.

When an [event handler content attribute](#) is removed, the user agent must set the corresponding [event handler](#) to null.



When an [event handler](#) *H* of an element or object *T* implementing the [EventTarget](#) interface is first set to a non-null value, the user agent must append an [event listener](#) to the list of [event listeners](#) associated with *T* with **type** set to the **event handler event type** corresponding to *H* and **callback** set to [the event handler processing algorithm](#) defined below. [\[DOM\]](#)

NOTE:

The **callback** is emphatically not the [event handler](#) itself. Every event handler ends up registering the same **callback** the algorithm defined below, which takes care of invoking the right callback, and processing the callback's return value.

NOTE:

This only happens the first time the [event handler](#)'s value is set. Since listeners are called in the order they were registered, the order of event listeners for a particular event type will always be first the event listeners registered with `addEventListener()` before the first time the [event handler](#) was set to a non-null value, then the callback to which it is currently set, if any, and finally the event listeners registered with `addEventListener()` after the first time the [event handler](#) was set to a non-null value.

EXAMPLE 626

This example demonstrates the order in which event listeners are invoked. If the button in this example is clicked by the user, the page will show four alerts, with the text "ONE", "TWO", "THREE", and "FOUR" respectively.

```
<button>Start Demo</button>
<script>
var button = document.getElementById('test');
button.addEventListener('click', function () { alert('ONE') }, false);
button.setAttribute('onclick', "alert('NOT CALLED')"); // event handler listener is registered here
button.addEventListener('click', function () { alert('THREE') }, false);
button.onclick = function () { alert('TWO'); };
button.addEventListener('click', function () { alert('FOUR') }, false);
</script>
```

NOTE:

The interfaces implemented by the event object do not influence whether an [event handler](#) is triggered or not.

The **event handler processing algorithm** for an [event handler](#) *H* and an [Event](#) object *E* is as follows:

1. Let *callback* be the result of [getting the current value of the event handler](#) *H*.
2. If *callback* is null, then abort these steps.
3. Process the [Event](#) object *E* as follows:

↪ If *E* is an [ErrorEvent](#) object and the [event handler IDL attribute](#)'s type is `OnErrorEventHandler`

[Invoke](#) *callback* with five arguments, the first one having the value of *E*'s [message](#) attribute, the second having the value of *E*'s [filename](#) attribute, the third having the value of *E*'s [lineno](#) attribute, the fourth having the value of *E*'s [colno](#) attribute, the fifth having the value of *E*'s [error](#) attribute, and with the [callback this value](#) set to *E*'s [currentTarget](#). Let *return value* be the callback's return value. [\[WEBIDL\]](#)

↪ Otherwise

[Invoke](#) *callback* with one argument, the value of which is the [Event](#) object *E*, with the [callback this value](#)

set to E 's [currentTarget](#). Let *return value* be the callback's return value. [\[WEBIDL\]](#)

In this step, **invoke** means to [invoke the Web IDL callback function](#).

If an exception gets thrown by the callback, end these steps and allow the exception to propagate. (It will propagate to the [DOM event dispatch logic](#), which will then [report the exception](#).)

4. Process *return value* as follows:

↪ If the event type is **mouseover**

↪ If the event type is **error** and E is an [ErrorEvent](#) object

If *return value* is a Web IDL boolean true value, then cancel the event.

↪ If the event type is **beforeunload**

NOTE:

The [event handler IDL attribute](#)'s type is `OnBeforeUnloadEventHandler`, and the *return value* will therefore have been coerced into either the value `null` or a `DOMString`.

If the *return value* is `null`, then cancel the event.

Otherwise, if the [Event](#) object E is a [BeforeUnloadEvent](#) object, and the [Event](#) object E 's [returnValue](#) attribute's value is the empty string, then set the [returnValue](#) attribute's value to *return value*.

↪ Otherwise

If *return value* is a Web IDL boolean false value, then cancel the event.



The [EventHandler](#) callback function type represents a callback used for event handlers. It is represented in Web IDL as follows:

```
[TreatNonObjectAsNull]
callback EventHandlerNonNull = any (Event event);
typedef EventHandlerNonNull? EventHandler;
```

NOTE:

In JavaScript, any [Function](#) object implements this interface.

EXAMPLE 627

For example, the following document fragment:

```
<body onload="alert(this)" onclick="alert(this)">
```

...leads to an alert saying "[object Window]" when the document is loaded, and an alert saying "[object HTMLBodyElement]" whenever the user clicks something in the page.

NOTE:

The return value of the function affects whether the event is canceled or not: as described above, if the return value is false, the event is canceled (except for mouseover events, where the return value has to be true to cancel the event). With beforeunload events, the value is instead used to determine whether or not to prompt about unloading the document.

For historical reasons, the [onerror](#) handler has different arguments:

```
[TreatNonObjectAsNull]
callback OnErrorHandlerNonNull = any ((Event or DOMString) event, optional DOMString
source, optional unsigned long lineno, optional unsigned long column, optional any error);
typedef OnErrorHandlerNonNull? OnErrorHandler;
```

Similarly, the [onbeforeunload](#) handler has a different return value:

```
[TreatNonObjectAsNull]
callback OnBeforeUnloadEventHandlerNonNull = DOMString? (Event event);
typedef OnBeforeUnloadEventHandlerNonNull? OnBeforeUnloadEventHandler;
```



An **internal raw uncompiled handler** is a tuple with the following information:

- An uncompiled script body
- A location where the script body originated, in case an error needs to be reported

When the user agent is to **get the current value of the event handler** *H*, it must run these steps:

1. If *H*'s value is an [internal raw uncompiled handler](#), run these substeps:
 1. If *H* is an element's [event handler](#), then let *element* be the element, and *document* be the element's [node document](#).
 - Otherwise, *H* is a [Window](#) object's [event handler](#): let *element* be null, and let *document* be the [Document](#) most recently associated with that [Window](#) object.
 2. If *document* does not have a [browsing context](#), or if [scripting is enabled](#) for *document*'s [browsing context](#), then return null.
 3. Let *body* be the uncompiled script body in the [internal raw uncompiled handler](#).
 4. Let *location* be the location where the script body originated, as given by the [internal raw uncompiled handler](#).
 5. If *element* is not null and *element* has a [form owner](#), let *form owner* be that [form owner](#). Otherwise, let *form owner* be null.
 6. Let *script settings* be the [environment settings object](#) created for the [Window](#) object with which *document* is currently associated.
 7. If *body* is not parsable as [FunctionBody](#) or if parsing detects an [early error](#), then follow these substeps:
 1. Set *H*'s value to null.
 2. [Report the error](#) for the appropriate [script](#) and with the appropriate position (line number and column number) given by *location*, using the [global object](#) specified by *script settings* as the target. If the error is still [not handled](#) after this, then the error may be reported to a developer console.
 3. Return null.
 8. If *body* begins with a [Directive Prologue](#) that contains a [Use Strict Directive](#) then let *strict* be true, otherwise let *strict* be false.
 9. Let *function* be the result of calling [FunctionCreate](#), with arguments:

kind

Normal

ParameterList

↪ If *H* is an [onerror](#) event handler of a [Window](#) object

Let the function have five arguments, named *event*, *source*, *lineno*, *colno*, and *error*.

↪ Otherwise

Let the function have a single argument called *event*.

Body

The result of parsing *body* above.

Scope

1. If *H* is an element's [event handler](#), then let *Scope* be the result of [NewObjectEnvironment](#)(*document*, the *global environment*).

Otherwise, *H* is a [Window](#) object's [event handler](#): let *Scope* be the *global environment*.

2. If *form owner* is not null, let *Scope* be [NewObjectEnvironment](#)(*form owner*, *Scope*).
3. If *element* is not null, let *Scope* be the [NewObjectEnvironment](#)(*element*, *Scope*).

Strict

The value of *strict*.

10. Set *H*'s value to *function*.

2. Return *H*'s value.

§ 7.1.5.2. Event handlers on elements, Document objects, and Window objects

The following are the [event handlers](#) (and their corresponding [event handler event types](#)) that must be supported by all [html elements](#), as both [event handler content attributes](#) and [event handler IDL attributes](#); and that must be supported by all [Document](#) and [Window](#) objects, as [event handler IDL attributes](#):

Event handler	Event handler event type
onabort	abort
oncancel	cancel
oncanplay	canplay
oncanplaythrough	canplaythrough
onchange	change
onclick	click
onclose	close
oncontextmenu	contextmenu
oncuechange	cuechange
ondblclick	dblclick
ondrag	drag
ondragend	dragend
ondragenter	dragenter
ondragexit	dragexit
ondragleave	dragleave
ondragover	dragover
ondragstart	dragstart
ondrop	drop
ondurationchange	durationchange
onemptied	emptied
onended	ended

Event handler	Event handler event type
oninput	input
oninvalid	invalid
onkeydown	keydown
onkeypress	keypress
onkeyup	keyup
onloadeddata	loadeddata
onloadedmetadata	loadedmetadata
onloadstart	loadstart
onmousedown	mousedown
onmouseenter	mouseenter
onmouseleave	mouseleave
onmousemove	mousemove
onmouseout	mouseout
onmouseover	mouseover
onmouseup	mouseup
onwheel	wheel
onpause	pause
onplay	play
onplaying	playing
onprogress	progress
onratechange	ratechange
onreset	reset
onseeked	seeked
onseeking	seeking
onselect	select
onshow	show
onstalled	stalled
onsubmit	submit
onsuspend	suspend
ontimeupdate	timeupdate
ontoggle	toggle
onvolumechange	volumechange
onwaiting	waiting



The following are the [event handlers](#) (and their corresponding [event handler event types](#)) that must be supported by all [html elements](#) other than `<body>` and `<frameset>` elements, as both [event handler content attributes](#) and [event handler IDL attributes](#); that must be supported by all [Document](#) objects, as [event handler IDL attributes](#); and that must be supported by all [Window](#) objects, as [event handler IDL attributes](#) on the [Window](#) objects themselves, and with corresponding [event handler content attributes](#) and [event handler IDL attributes](#) exposed on all `<body>` and `<frameset>` elements that are owned by that [Window](#) object's [Documents](#):

Event handler	Event handler event type
onblur	blur

<u>Event handler</u>	<u>Event handler event type</u>
onerror	error
onfocus	focus
onload	load
onresize	resize
onscroll	scroll



The following are the [event handlers](#) (and their corresponding [event handler event types](#)) that must be supported by [Window](#) objects, as [event handler IDL attributes](#) on the [Window](#) objects themselves, and with corresponding [event handler content attributes](#) and [event handler IDL attributes](#) exposed on all [<body>](#) and [<frameset>](#) elements that are owned by that [Window](#) object's [Documents](#):

<u>Event handler</u>	<u>Event handler event type</u>
onafterprint	afterprint
onbeforeprint	beforeprint
onbeforeunload	beforeunload
onhashchange	hashchange
onlanguagechange	languagechange
onmessage	message
onoffline	offline
ononline	online
onpagehide	pagehide
onpageshow	pageshow
onrejectionhandled	rejectionhandled
onpopstate	popstate
onstorage	storage
onunhandledrejection	unhandledrejection
onunload	unload



The following are the [event handlers](#) (and their corresponding [event handler event types](#)) that must be supported by all [html elements](#), as both [event handler content attributes](#) and [event handler IDL attributes](#) and that must be supported by all [Document](#) objects, as [event handler IDL attributes](#):

<u>Event handler</u>	<u>Event handler event type</u>
oncut	cut
oncopy	copy
onpaste	paste



The following are the [event handlers](#) (and their corresponding [event handler event types](#)) that must be supported on [Document](#) objects as [event handler IDL attributes](#):

<u>Event handler</u>	<u>Event handler event type</u>
onreadystatechange	readystatechange

§ 7.1.5.2.1. IDL DEFINITIONS

```
[NoInterfaceObject]
interface GlobalEventHandlers {
  attribute EventHandler onabort;
  attribute EventHandler onblur;
  attribute EventHandler oncancel;
  attribute EventHandler oncanplay;
  attribute EventHandler oncanplaythrough;
  attribute EventHandler onchange;
  attribute EventHandler onclick;
  attribute EventHandler onclose;
  attribute EventHandler oncontextmenu;
  attribute EventHandler oncuechange;
  attribute EventHandler ondblclick;
  attribute EventHandler ondrag;
  attribute EventHandler ondragend;
  attribute EventHandler ondragenter;
  attribute EventHandler ondragexit;
  attribute EventHandler ondragleave;
  attribute EventHandler ondragover;
  attribute EventHandler ondragstart;
  attribute EventHandler ondrop;
  attribute EventHandler ondurationchange;
  attribute EventHandler onemptied;
  attribute EventHandler onended;
  attribute OnErrorEventHandler onerror;
  attribute EventHandler onfocus;
  attribute EventHandler oninput;
  attribute EventHandler oninvalid;
  attribute EventHandler onkeydown;
  attribute EventHandler onkeypress;
  attribute EventHandler onkeyup;
  attribute EventHandler onload;
  attribute EventHandler onloadeddata;
  attribute EventHandler onloadedmetadata;
  attribute EventHandler onloadstart;
  attribute EventHandler onmousedown;
  [LenientThis] attribute EventHandler onmouseenter;
  [LenientThis] attribute EventHandler onmouseleave;
  attribute EventHandler onmousemove;
  attribute EventHandler onmouseout;
  attribute EventHandler onmouseover;
  attribute EventHandler onmouseup;
  attribute EventHandler onwheel;
  attribute EventHandler onpause;
  attribute EventHandler onplay;
  attribute EventHandler onplaying;
  attribute EventHandler onprogress;
  attribute EventHandler onratechange;
  attribute EventHandler onreset;
  attribute EventHandler onresize;
  attribute EventHandler onscroll;
  attribute EventHandler onseeked;
  attribute EventHandler onseeking;
  attribute EventHandler onselect;
  attribute EventHandler onshow;
  attribute EventHandler onstalled;
  attribute EventHandler onsubmit;
  attribute EventHandler onsuspend;
  attribute EventHandler ontimeupdate;
  attribute EventHandler ontoggle;
  attribute EventHandler onvolumechange;
  attribute EventHandler onwaiting;
};
```

```
[NoInterfaceObject]
interface WindowEventHandlers {
  attribute EventHandler onafterprint;
  attribute EventHandler onbeforeprint;
  attribute OnBeforeUnloadEventHandler onbeforeunload;
  attribute EventHandler onhashchange;
  attribute EventHandler onlanguagechange;
  attribute EventHandler onmessage;
  attribute EventHandler onoffline;
  attribute EventHandler ononline;
  attribute EventHandler onpagehide;
  attribute EventHandler onpageshow;
  attribute EventHandler onrejectionhandled;
  attribute EventHandler onpopstate;
  attribute EventHandler onstorage;
  attribute EventHandler onunhandledrejection;
  attribute EventHandler onunload;
};
```

```
[NoInterfaceObject]
interface DocumentAndElementEventHandlers {
  attribute EventHandler oncopy;
  attribute EventHandler oncut;
  attribute EventHandler onpaste;
};
```

§ 7.1.5.3. Event firing

Certain operations and methods are defined as firing events on elements. For example, the [click\(\)](#) method on the [HTMLElement](#) interface is defined as firing a click event on the element. [\[UIEVENTS\]](#)

Firing a simple event named *e* means that a [trusted](#) event with the name *e*, which does not bubble (except where otherwise stated) and is not cancelable (except where otherwise stated), and which uses the [Event](#) interface, must be created and [dispatched](#) at the given target.

Firing a synthetic mouse event named *e* means that an event with the name *e*, which is [trusted](#) (except where otherwise stated), does not bubble (except where otherwise stated), is not cancelable (except where otherwise stated), and which uses the [MouseEvent](#) interface, must be created and [dispatched](#) at the given target. The event object must have its [screenX](#), [screenY](#), {[MouseEvent/clientX](#)}, [clientY](#), and [button](#) attributes initialized to 0, its [ctrlKey](#), [shiftKey](#), [altKey](#), and [metaKey](#) attributes initialized according to the current state of the key input device, if any (false for any keys that are not available), its [detail](#) attribute initialized to 1, its [relatedTarget](#) attribute initialized to null (except where otherwise stated), and its [view](#) attribute initialized to the [Window](#) object of the [Document](#) object of the given target node, if any, or else null. The [getModifierState\(\)](#) method on the object must return values appropriately describing the state of the key input device at the time the event is created.

Firing a click event means [firing a synthetic mouse event named click](#), which bubbles and is cancelable.

The default action of these events is to do nothing except where otherwise stated.

§ 7.1.5.4. Events and the Window object

When an event is dispatched at a DOM node in a Document in a [browsing context](#), if the event is not a load event, the user agent must act as if, for the purposes of [event dispatching](#), the Window object is the parent of the Document object. [\[DOM\]](#)

§ 7.2. Base64 utility methods

The [atob\(\)](#) and [btoa\(\)](#) methods allow authors to transform content to and from the base64 encoding.

```
[NoInterfaceObject, Exposed=(Window, Worker)]
interface WindowBase64 {
  DOMString btoa(DOMString btoa);
  DOMString atob(DOMString atob);
};
Window implements WindowBase64;
WorkerGlobalScope implements WindowBase64;
```

NOTE:

In these APIs, for mnemonic purposes, the "b" can be considered to stand for "binary", and the "a" for "ASCII". In practice, though, for primarily historical reasons, both the input and output of these functions are Unicode strings.

This definition is non-normative. Implementation requirements are given below this definition.

result = **window** . **btoa**(**data**)

Takes the input data, in the form of a Unicode string containing only characters in the range U+0000 to U+00FF, each representing a binary byte with values 0x00 to 0xFF respectively, and converts it to its base64 representation, which it returns.

Throws an `InvalidCharacterError` exception if the input string contains any out-of-range characters.

result = **window** . **atob**(**data**)

Takes the input data, in the form of a Unicode string containing base64-encoded binary data, decodes it, and returns a string consisting of characters in the range U+0000 to U+00FF, each representing a binary byte with values 0x00 to 0xFF respectively, corresponding to that binary data.

Throws an `InvalidCharacterError` exception if the input string is not valid base64 data.

The **btoa()** method must throw an `InvalidCharacterError` exception if the method's first argument contains any character whose code point is greater than U+00FF. Otherwise, the user agent must convert that argument to a sequence of octets whose *n*th octet is the eight-bit representation of the code point of the *n*th character of the argument, and then must apply the base64 algorithm to that sequence of octets, and return the result. [\[RFC4648\]](#)

The **atob()** method must run the following steps to parse the string passed in the method's first argument:

1. Let *input* be the string being parsed.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
3. Remove all [space characters](#) from *input*.
4. If the length of *input* divides by 4 leaving no remainder, then: if *input* ends with one or two U+003D EQUALS SIGN (=) characters, remove them from *input*.
5. If the length of *input* divides by 4 leaving a remainder of 1, throw an `InvalidCharacterError` exception and abort these steps.
6. If *input* contains a character that is not in the following list of characters and character ranges, throw an `InvalidCharacterError` exception and abort these steps:
 - U+002B PLUS SIGN (+)
 - U+002F SOLIDUS (/)
 - [Alphanumeric ASCII characters](#)
7. Let *output* be a string, initially empty.
8. Let *buffer* be a buffer that can have bits appended to it, initially empty.
9. While *position* does not point past the end of *input*, run these substeps:
 1. Find the character pointed to by *position* in the first column of the following table. Let *n* be the number given in the second cell of the same row.

Character	Number

Character	Number
A	0
B	1
C	2
D	3
E	4
F	5
G	6
H	7
I	8
J	9
K	10
L	11
M	12
N	13
O	14
P	15
Q	16
R	17
S	18
T	19
U	20
V	21
W	22
X	23
Y	24
Z	25
a	26
b	27
c	28
d	29
e	30
f	31
g	32
h	33
i	34
j	35
k	36
l	37
m	38
n	39
o	40
p	41
q	42

Character	Number
r	43
s	44
t	45
u	46
v	47
w	48
x	49
y	50
z	51
0	52
1	53
2	54
3	55
4	56
5	57
6	58
7	59
8	60
9	61
+	62
/	63

2. Append to *buffer* the six bits corresponding to *number*, most significant bit first.
3. If *buffer* has accumulated 24 bits, interpret them as three 8-bit big-endian numbers. Append the three characters with code points equal to those numbers to *output*, in the same order, and then empty *buffer*.
4. Advance *position* by one character.
10. If *buffer* is not empty, it contains either 12 or 18 bits. If it contains 12 bits, discard the last four and interpret the remaining eight as an 8-bit big-endian number. If it contains 18 bits, discard the last two and interpret the remaining 16 as two 8-bit big-endian numbers. Append the one or two characters with code points equal to those one or two numbers to *output*, in the same order.

NOTE:

The discarded bits mean that, for instance, `atob("YQ")` and `atob("YR")` both return "`<a>`".

11. Return *output*.

§ 7.3. Dynamic markup insertion

NOTE:

APIs for dynamically inserting markup into the document interact with the parser, and thus their behavior varies depending on whether they are used with [HTML documents](#) (and the [HTML parser](#)) or XHTML in [XML documents](#) (and the [XML parser](#)).

§ 7.3.1. Opening the input stream

The `open()` method comes in several variants with different numbers of arguments.

This definition is non-normative. Implementation requirements are given below this definition.

`document = document . open( [ type [, replace ] ] )`

Causes the [Document](#) to be replaced in-place, as if it was a new [Document](#) object, but reusing the previous object, which is then returned.

If the `type` argument is omitted or has the value `"text/html"`, then the resulting [Document](#) has an HTML parser associated with it, which can be given data to parse using [document.write\(\)](#). Otherwise, all content passed to [document.write\(\)](#) will be parsed as plain text.

If the `replace` argument is present and has the value `"replace"`, the existing entries in the session history for the [Document](#) object are removed.

The method has no effect if the [Document](#) is still being parsed.

Throws an `"InvalidStateError"` [DOMException](#) if the [Document](#) is an [XML document](#).

`window = document . open( url, name, features [, replace ] )`

Works like the [window.open\(\)](#) method.

[Document](#) objects have an **ignore-opens-during-unload counter**, which is used to prevent scripts from invoking the [document.open\(\)](#) method (directly or indirectly) while the document is [being unloaded](#). Initially, the counter must be set to zero.

When called with two arguments (or fewer), the [document.open\(\)](#) method must act as follows:

1. If the [Document](#) object is an [XML document](#), then throw an `"InvalidStateError"` [DOMException](#) and abort these steps.
2. If the [Document](#) object is not an [active document](#), then abort these steps.
3. If the [origin](#) of the [Document](#) is not equal to the [origin](#) of the [responsible document](#) specified by the [entry settings object](#), throw a `"SecurityError"` [DOMException](#) and abort these steps.
4. Let `type` be the value of the first argument.
5. If the second argument is an [ASCII case-insensitive](#) match for the value `"replace"`, then let `replace` be true.

Otherwise, if the [browsing context](#)'s [session history](#) contains only one [Document](#), and that was the [about:blank Document](#) created when the [browsing context](#) was created, and that [Document](#) has never had the [unload a document](#) algorithm invoked on it (e.g., by a previous call to [document.open\(\)](#)), then let `replace` be true.

Otherwise, let `replace` be false.

6. If the [Document](#) has an [active parser](#) whose [script nesting level](#) is greater than zero, then the method does nothing. Abort these steps and return the [Document](#) object on which the method was invoked.

NOTE:

This basically causes [document.open\(\)](#) to be ignored when it's called in an inline script found during parsing, while still letting it have an effect when called from a non-parser task such as a timer callback or event handler.

7. Similarly, if the [Document](#)'s [ignore-opens-during-unload counter](#) is greater than zero, then the method does nothing. Abort these steps and return the [Document](#) object on which the method was invoked.

NOTE:

This basically causes [document.open\(\)](#) to be ignored when it's called from a [beforeunload](#) pagehide, or unload event handler while the [Document](#) is being unloaded.

8. Set the [Document](#)'s [salvageable](#) state to false.
9. [Prompt to unload](#) the [Document](#) object. If the user [refused to allow the document to be unloaded](#), then abort these steps and return the [Document](#) object on which the method was invoked.
10. [Unload](#) the [Document](#) object, with the `recycle` parameter set to true.

11. [Abort](#) the [Document](#).
 12. Unregister all event listeners registered on the [Document](#) node and its descendants.
 13. Remove any [tasks](#) associated with the [Document](#) in any [task source](#).
 14. Remove all child nodes of the document, without firing any mutation events.
 15. Call the JavaScript [InitializeHostDefinedRealm\(\)](#) abstract operation with the following customizations:
 - For the global object, create a new [Window](#) object *window*.
 - For the global *this* value, use the current [browsing context](#)'s associated [WindowProxy](#).
 - Let *realm execution context* be the created [JavaScript execution context](#).
 16. Set *window*'s [associated Document](#) to the [Document](#).
 17. [Set up a browsing context environment settings object](#) with *realm execution context*.
 18. Replace the [Document](#)'s singleton objects with new instances of those objects, created in *window*'s [Realm](#). (This includes in particular the [History](#), [ApplicationCache](#) and [Navigator](#) objects, the various [BarProp](#) objects, the two Storage objects, the various [HTMLCollection](#) objects, and objects defined by other specifications, like Selection. It also includes all the Web IDL prototypes in the JavaScript binding, including the [Document](#) object's prototype.)
 19. Change the [document's character encoding](#) to UTF-8.
 20. If the [Document](#) is [ready for post-load tasks](#), then set the [Document](#) object's [reload override flag](#) and set the [Document's reload override buffer](#) to the empty string.
 21. Set the [Document's salvageable](#) state back to true.
 22. Change the document's [URL](#) to the [URL](#) of the [responsible document](#) specified by the [entry settings object](#).
 23. If the [Document's iframe load in progress](#) flag is set, set the [Document's mute iframe load](#) flag.
 24. Create a new [HTML parser](#) and associate it with the document. This is a **script-created parser** (meaning that it can be closed by the [document.open\(\)](#) and [document.close\(\)](#) methods, and that the tokenizer will wait for an explicit call to [document.close\(\)](#) before emitting an end-of-file token). The encoding [confidence](#) is *irrelevant*.
 25. Set the [current document readiness](#) of the document to "loading".
 26. If *type* is an [ASCII case-insensitive](#) match for the string "replace", then, for historical reasons, set it to the string "text/html".
 Otherwise:
 If the *type* string contains a U+003B SEMICOLON character (;), remove the first such character and all characters from it up to the end of the string.
[Strip leading and trailing whitespace](#) from *type*.
 27. If *type* is *not* now an [ASCII case-insensitive](#) match for the string "text/html", then act as if the tokenizer had emitted a start tag token with the tag name "pre" followed by a single U+000A LINE FEED (LF) character, then switch the [HTML parser's](#) tokenizer to the [§8.2.4.7 PLAINTEXT state](#).
 28. Remove all the entries in the [browsing context's session history](#) after the [current entry](#). If the [current entry](#) is the last entry in the session history, then no entries are removed.
- NOTE:**
 This [doesn't necessarily have to affect](#) the user agent's user interface.
29. Remove any [tasks](#) queued by the [history traversal task source](#) that are associated with any [Document](#) objects in the [top-level browsing context's document family](#).

30. Remove any earlier entries that share the same [Document](#).
31. If `replace` is false, then add a new entry, just before the last entry, and associate with the new entry the text that was parsed by the previous parser associated with the [Document](#) object, as well as the state of the document at the start of these steps. This allows the user to step backwards in the session history to see the page before it was blown away by the `document.open()` call. This new entry does not have a [Document](#) object, so a new one will be created if the session history is traversed to that entry.
32. Set the [Document](#)'s `fired unload` flag to false. (It could have been set to true during the unload step above.)
33. Finally, set the [insertion point](#) to point at just before the end of the [input stream](#) (which at this point will be empty).
34. Return the [Document](#) on which the method was invoked.

NOTE:

The `document.open()` method does not affect whether a [Document](#) is [ready for post-load tasks](#) or [completely loaded](#).

When called with four arguments, the `open()` method on the [Document](#) object must call the `open()` method on the [Window](#) object of the [Document](#) object, with the same arguments as the original call to the `open()` method, and return whatever that method returned. If the [Document](#) object has no [Window](#) object, then the method must throw an `"InvalidAccessError"` [DOMException](#).

§ 7.3.2. Closing the input stream

This definition is non-normative. Implementation requirements are given below this definition.

`document.close()`

Closes the input stream that was opened by the `document.open()` method.

Throws an `InvalidStateError` exception if the [Document](#) is an [XML document](#).

The `close()` method must run the following steps:

1. If the [Document](#) object is not flagged as an [HTML document](#), throw an `InvalidStateError` exception and abort these steps.
2. If there is no [script-created parser](#) associated with the document, then abort these steps.
3. Insert an [explicit "EOF" character](#) at the end of the parser's [input stream](#).
4. If there is a [pending parsing-blocking script](#), then abort these steps.
5. Run the tokenizer, processing resulting tokens as they are emitted, and stopping when the tokenizer reaches the [explicit "EOF" character](#) or [spins the event loop](#).

§ 7.3.3. `document.write()`

This definition is non-normative. Implementation requirements are given below this definition.

`document.write(text...)`

In general, adds the given string(s) to the Document's input stream.

⚠Warning! This method has very idiosyncratic behavior. In some cases, this method can affect the state of the [HTML parser](#) while the parser is running, resulting in a DOM that does not correspond to the source of the document (e.g., if the string written is the string "<plaintext>" or "<!--"). In other cases, the call can clear the current page first, as if `document.open()` had been called. In yet more cases, the method is simply ignored, or throws an exception. To make matters worse, the exact behavior of this method can in some cases be dependent on network latency, which can lead to failures that are very hard to debug. **For all these reasons, use of this method is strongly discouraged.**

This method throws an `InvalidStateError` exception when invoked on [XML documents](#).

Document objects have an **ignore-destructive-writes counter**, which is used in conjunction with the processing of `<script>` elements to prevent external scripts from being able to use [document.write\(\)](#) to blow away the document by implicitly calling `document.open()`. Initially, the counter must be set to zero.

The **document.write(...)** method must act as follows:

1. If the method was invoked on an [XML document](#), throw an `InvalidStateError` exception and abort these steps.
2. If the Document object is not an [active document](#), then abort these steps.
3. If the [insertion point](#) is undefined and either the Document's [ignore-opens-during-unload counter](#) is greater than zero or the Document's [ignore-destructive-writes counter](#) is greater than zero, abort these steps.
4. If the [insertion point](#) is undefined, call the `open()` method on the document object (with no arguments). If the user [refused to allow the document to be unloaded](#), then abort these steps. Otherwise, the [insertion point](#) will point at just before the end of the (empty) [input stream](#).
5. Insert the string consisting of the concatenation of all the arguments to the method into the [input stream](#) just before the [insertion point](#).
6. If the Document object's [reload override flag](#) is set, then append the string consisting of the concatenation of all the arguments to the method to the Document's [reload override buffer](#).
7. If there is no [pending parsing-blocking script](#), have the [HTML parser](#) process the characters that were inserted, one at a time, processing resulting tokens as they are emitted, and stopping when the tokenizer reaches the insertion point or when the processing of the tokenizer is aborted by the tree construction stage (this can happen if a script end tag token is emitted by the tokenizer).

NOTE:

If the [document.write\(\)](#) method was called from script executing inline (i.e., executing because the parser parsed a set of script tags), then this is a [reentrant invocation of the parser](#).

8. Finally, return from the method.

§ 7.3.4. document.writeln()

This definition is non-normative. Implementation requirements are given below this definition.

`document.writeln(text...)`

Adds the given string(s) to the Document's input stream, followed by a newline character. If necessary, calls the `open()` method implicitly first.

This method throws an `InvalidStateError` exception when invoked on [XML documents](#).

The **document.writeln(...)** method, when invoked, must act as if the [document.write\(\)](#) method had been invoked with the same argument(s), plus an extra argument consisting of a string containing a single line feed character (U+000A).

§ 7.4. Timers

The `setTimeout()` and `setInterval()` methods allow authors to schedule timer-based callbacks.

```
[NoInterfaceObject, Exposed=(Window,Worker)]
interface WindowTimers {
  long setTimeout((Function or DOMString) handler, optional long timeout = 0, any... arguments);
  void clearTimeout(optional long handle = 0);
  long setInterval((Function or DOMString) handler, optional long timeout = 0, any...
arguments);
  void clearInterval(optional long handle = 0);
};
Window implements WindowTimers;
WorkerGlobalScope implements WindowTimers;
```

This definition is non-normative. Implementation requirements are given below this definition.

`handle = window.setTimeout( handler [, timeout [, arguments ... ] ] )`

Schedules a timeout to run *handler* after *timeout* milliseconds. Any *arguments* are passed straight through to the *handler*.

`handle = window.setTimeout( code [, timeout ] )`

Schedules a timeout to compile and run *code* after *timeout* milliseconds.

`window.clearTimeout( handle )`

Cancels the timeout set with `setTimeout()` or `setInterval()` identified by *handle*.

`handle = window.setInterval( handler [, timeout [, arguments ... ] ] )`

Schedules a timeout to run *handler* every *timeout* milliseconds. Any *arguments* are passed straight through to the *handler*.

`handle = window.setInterval( code [, timeout ] )`

Schedules a timeout to compile and run *code* every *timeout* milliseconds.

`window.clearInterval( handle )`

Cancels the timeout set with `setInterval()` or `setTimeout()` identified by *handle*.

NOTE:

Timers can be nested; after five such nested timers, however, the interval is forced to be at least four milliseconds.

NOTE:

This API does not guarantee that timers will run exactly on schedule. Delays due to CPU load, other tasks, etc, are to be expected.

Objects that implement the `WindowTimers` interface have a **list of active timers**. Each entry in this lists is identified by a number, which must be unique within the list for the lifetime of the object that implements the `WindowTimers` interface.



The `setTimeout()` method must return the value returned by the [timer initialization steps](#), passing them the method's arguments, the object on which the method for which the algorithm is running is implemented (a `Window` or `WorkerGlobalScope` object) as the *method context*, and the *repeat* flag set to false.

The `setInterval()` method must return the value returned by the [timer initialization steps](#), passing them the method's arguments, the object on which the method for which the algorithm is running is implemented (a `Window` or `WorkerGlobalScope` object) as the *method context*, and the *repeat* flag set to true.

The `clearTimeout()` and `clearInterval()` methods must clear the entry identified as *handle* from the [list of active timers](#) of the `WindowTimers` object on which the method was invoked, if any, where *handle* is the argument passed to the

method. (If *handle* does not identify an entry in the [list of active timers](#) of the WindowTimers object on which the method was invoked, the method does nothing.)

NOTE:

Because `clearTimeout()` and `clearInterval()` clear entries from the same list, either method can be used to clear timers created by `setTimeout()` or `setInterval()`.



The **timer initialization steps**, which are invoked with some method arguments, a *method context*, a *repeat* flag which can be true or false, and optionally (and only if the *repeat* flag is true) a *previous handle*, are as follows:

1. Let *method context proxy* be *method context* if that is a WorkerGlobalScope object, or else the WindowProxy that corresponds to *method context*.
2. If *previous handle* was provided, let *handle* be *previous handle*; otherwise, let *handle* be a user-agent-defined integer that is greater than zero that will identify the timeout to be set by this call in the [list of active timers](#).
3. If *previous handle* was not provided, add an entry to the [list of active timers](#) for *handle*.
4. Let *callerRealm* be the current Realm Record, and *calleeRealm* be *method context*'s [JavaScript realm](#).
5. Let *task* be a [task](#) that runs the following substeps:

1. If the entry for *handle* in the [list of active timers](#) has been cleared, then abort this [task](#)'s substeps.
2. Run the appropriate set of steps from the following list:

↪ **If the first method argument is a Function**

[Invoke](#) the Function. Use the third and subsequent method arguments (if any) as the arguments for invoking the Function. Use *method context proxy* as the [Callback this value](#). [\[ECMA-262\]](#)

↪ **Otherwise**

1. Perform [HostEnsureCanCompileStrings](#)(*callerRealm*, *calleeRealm*). If this throws an exception, [report the exception](#).
2. Let *script source* be the first method argument.
3. Let *script language* be JavaScript.
4. Let *settings object* be *method context*'s [environment settings object](#).
5. [Create a script](#) using *script source* as the script source, the [URL](#) where *script source* can be found, *scripting language* as the scripting language, and *settings object* as the [environment settings object](#).
3. If the *repeat* flag is true, then call [timer initialization steps](#) again, passing them the same method arguments, the same *method context*, with the *repeat* flag still set to true, and with the *previous handle* set to *handler*.
6. Let *timeout* be the second method argument.
7. If the currently running [task](#) is a task that was created by this algorithm, then let *nesting level* be the [task](#)'s [timer nesting level](#). Otherwise, let *nesting level* be zero.
8. If *nesting level* is greater than 5, and *timeout* is less than 4, then increase *timeout* to 4.
9. Increment *nesting level* by one.
10. Let *task*'s **timer nesting level** be *nesting level*.
11. Return *handle*, and then continue running this algorithm [in parallel](#).
12. If *method context* is a Window object, wait until the Document associated with *method context* has been [fully active](#) for

a further *timeout* milliseconds (not necessarily consecutively).

Otherwise, *method context* is a WorkerGlobalScope object; wait until *timeout* milliseconds have passed with the worker not suspended (not necessarily consecutively).

13. Wait until any invocations of this algorithm that had the same *method context*, that started before this one, and whose *timeout* is equal to or less than this one's, have completed.

NOTE:

Argument conversion as defined by Web IDL (for example, invoking `toString()` methods on objects passed as the first argument) happens in the algorithms defined in Web IDL, before this algorithm is invoked.

EXAMPLE 628

So for example, the following rather silly code will result in the log containing "ONE TWO ":

```
var log = '';function logger(s) { log += s + ' '; }

setTimeout({ toString: function () {
  setTimeout("logger('ONE')", 100);
  return "logger('TWO')";
} }, 100);
```

14. Optionally, wait a further user-agent defined length of time.

NOTE:

This is intended to allow user agents to pad timeouts as needed to optimize the power usage of the device. For example, some processors have a low-power mode where the granularity of timers is reduced; on such platforms, user agents can slow timers down to fit this schedule instead of requiring the processor to use the more accurate mode with its associated higher power usage.

15. [Queue](#) the [task](#) *task*.

NOTE:

Once the task has been processed, if the *repeat* flag is false, it is safe to remove the entry for *handle* from the [list of active timers](#) (there is no way for the entry's existence to be detected past this point, so it does not technically matter one way or the other).

The [task source](#) for these [tasks](#) is the **timer task source**.

EXAMPLE 629

To run tasks of several milliseconds back to back without any delay, while still yielding back to the browser to avoid starving the user interface (and to avoid the browser killing the script for hogging the CPU), simply queue the next timer before performing work:

```
function doExpensiveWork() {var done = false;
// ...
// this part of the function takes up to five milliseconds
// set done to true if we're done
// ...
return done;
}

function rescheduleWork() {
var handle = setTimeout(rescheduleWork, 0); // preschedule next iteration
if (doExpensiveWork())
  clearTimeout(handle); // clear the timeout if we don't need it
}

function scheduleWork() {
setTimeout(rescheduleWork, 0);
}

scheduleWork(); // queues a task to do lots of work
```

§ **7.5. User prompts**§ **7.5.1. Simple dialogs**

This definition is non-normative. Implementation requirements are given below this definition.

`window . alert(message)`

Displays a modal alert with the given message, and waits for the user to dismiss it.

`result = window . confirm(message)`

Displays a modal OK/Cancel prompt with the given message, waits for the user to dismiss it, and returns true if the user clicks OK and false if the user clicks Cancel.

`result = window . prompt(message [, default])`

Displays a modal text field prompt with the given message, waits for the user to dismiss it, and returns the value that the user entered. If the user cancels the prompt, then returns null instead. If the second argument is present, then the given value is used as a default.

NOTE:

Logic that depends on [tasks](#) or [microtasks](#), such as [media elements](#) loading their [media data](#), are stalled when these methods are invoked.

To **optionally truncate a simple dialog string** `s`, return either `s` itself or some string derived from `s` that is shorter. User agents should not provide UI for displaying the elided portion of `s`, as this makes it too easy for abusers to create dialogs of the form "Important security alert! Click 'Show More' for full details!".

NOTE:

For example, a user agent might want to only display the first 100 characters of a message. Or, a user agent might replace the middle of the string with "...". These types of modifications can be useful in limiting the abuse potential of unnaturally large, trustworthy-looking system dialogs.

The **`alert(message)`** method, when invoked, must run the following steps:

1. If the [event loop](#)'s [termination nesting level](#) is non-zero, optionally abort these steps.

2. If the [active sandboxing flag set](#) of the [active document](#) of the [responsible browsing context](#) specified by the [incumbent settings object](#) has the [sandboxed modals flag](#) set, then abort these steps.
3. Optionally, abort these steps. (For example, the user agent might give the user the option to ignore all alerts, and would thus abort at this step whenever the method was invoked.)
4. If the method was invoked with no arguments, then let *message* be the empty string; otherwise, let *message* be the method's first argument.
5. Show the given *message* to the user.
6. Optionally, [pause](#) while waiting for the user to acknowledge the message.

The **confirm(*message*)** method, when invoked, must run the following steps:

1. If the [event loop's termination nesting level](#) is non-zero, optionally abort these steps, returning false.
2. If the [active sandboxing flag set](#) of the [active document](#) of the [responsible browsing context](#) specified by the [incumbent settings object](#) has the [sandboxed modals flag](#) set, then return false and abort these steps.
3. Optionally, return false and abort these steps. (For example, the user agent might give the user the option to ignore all prompts, and would thus abort at this step whenever the method was invoked.)
4. Set *message* to the result of [optionally truncating](#) *message*.
5. Show *message* to the user, and ask the user to respond with a positive or negative response.
6. [Pause](#) until the user responds either positively or negatively.
7. If the user responded positively, return true; otherwise, the user responded negatively: return false.

The **prompt(*message*, *default*)** method, when invoked, must run the following steps:

1. If the [event loop's termination nesting level](#) is non-zero, optionally abort these steps, returning null.
2. If the [active sandboxing flag set](#) of the [active document](#) of the [responsible browsing context](#) specified by the [incumbent settings object](#) has the [sandboxed modals flag](#) set, then return null and abort these steps.
3. Optionally, return null and abort these steps. (For example, the user agent might give the user the option to ignore all prompts, and would thus abort at this step whenever the method was invoked.)
4. Set *message* to the result of [optionally truncating](#) *message*.
5. Set *default* to the result of [optionally truncating](#) *default*.
6. Show *message* to the user, and ask the user to either respond with a string value or abort. The response must be defaulted to the value given by *default*.
7. [Pause](#) while waiting for the user's response.
8. If the user aborts, then return null; otherwise, return the string that the user responded with.

§ 7.5.2. Printing

This definition is non-normative. Implementation requirements are given below this definition.

`window.print()`

Prompts the user to print the page.

When the **print()** method is invoked, if the Document is [ready for post-load tasks](#), then the user agent must run the [printing steps in parallel](#). Otherwise, the user agent must only set the **print when loaded** flag on the [Document](#).

User agents should also run the [printing steps](#) whenever the user asks for the opportunity to [obtain a physical form](#) (e.g., printed copy), or the representation of a physical form (e.g., PDF copy), of a document.

The **printing steps** are as follows:

1. The user agent may display a message to the user or abort these steps (or both).

EXAMPLE 630

For instance, a kiosk browser could silently ignore any invocations of the `print()` method.

EXAMPLE 631

For instance, a browser on a mobile device could detect that there are no printers in the vicinity and display a message saying so before continuing to offer a "save to PDF" option.

2. If the [active sandboxing flag set](#) of the [active document](#) of the [responsible browsing context](#) specified by the [incumbent settings object](#) has the [sandboxed modals flag set](#), then abort these steps.

NOTE:

If the printing dialog is blocked by a Document's sandbox, then neither the beforeprint nor afterprint events will be fired.

3. The user agent must [fire a simple event](#) named beforeprint at the Window object of the Document that is being printed, as well as any [nested browsing contexts](#) in it.

EXAMPLE 632

The beforeprint event can be used to annotate the printed copy, for instance adding the time at which the document was printed.

4. The user agent should offer the user the opportunity to [obtain a physical form](#) (or the representation of a physical form) of the document. The user agent may wait for the user to either accept or decline before returning; if so, the user agent must [pause](#) while the method is waiting. Even if the user agent doesn't wait at this point, the user agent must use the state of the relevant documents as they are at this point in the algorithm if and when it eventually creates the alternate form.
5. The user agent must [fire a simple event](#) named afterprint at the Window object of the Document that is being printed, as well as any [nested browsing contexts](#) in it.

EXAMPLE 633

The afterprint event can be used to revert annotations added in the earlier event, as well as showing post-printing UI. For instance, if a page is walking the user through the steps of applying for a home loan, the script could automatically advance to the next step after having printed a form or other.

§ 7.5.3. Dialogs implemented using separate documents with `showModalDialog()`

This feature is in the process of being removed from the Web platform. (This is a long process that takes many years.) Using the `showModalDialog()` API at this time is highly discouraged.

The `showModalDialog(url, argument)` method, when invoked, must cause the user agent to run the following steps:

1. [Parse](#) *url* relative to the [API base URL](#) specified by the [entry settings object](#).
If this fails, then throw a `"SyntaxError" DOMException` and abort these steps.
2. If the [event loop's termination nesting level](#) is non-zero, optionally abort these steps, returning the empty string.
3. If the user agent is configured such that this invocation of `showModalDialog()` is somehow disabled, then return the empty string and abort these steps.

NOTE:

User agents are expected to disable this method in certain cases to avoid user annoyance (e.g., as part of their popup blocker feature). For instance, a user agent could require that a site be safelisted before enabling this method, or the user agent could be configured to only allow one modal dialog at a time.

4. If the [active sandboxing flag set](#) of the [active document](#) of the [responsible browsing context](#) specified by the [incumbent settings object](#) has either the [sandboxed auxiliary navigation browsing context flag](#) or [sandboxed modals flag](#) set, then return the empty string and abort these steps.
5. Let *incumbent origin* be the [origin](#) specified by the [incumbent settings object](#) at the time the `showModalDialog()` method was called.
6. Let *the list of background browsing contexts* be a list of all the browsing contexts that:
 - are part of the same [unit of related browsing contexts](#) as the browsing context of the Window object on which the `showModalDialog()` method was called, and that
 - have an [active document](#) whose [origin](#) is the same as *incumbent origin*,
 ...as well as any browsing contexts that are nested inside any of the browsing contexts matching those conditions.
7. Disable the user interface for all the browsing contexts in *the list of background browsing contexts*. This should prevent the user from navigating those browsing contexts, causing events to be sent to those browsing context, or editing any content in those browsing contexts. However, it does not prevent those browsing contexts from receiving events from sources other than the user, from running scripts, from running animations, and so forth.
8. Create a new [auxiliary browsing context](#), with the [opener browsing context](#) being the browsing context of the Window object on which the `showModalDialog()` method was called. The new auxiliary browsing context has no name.

NOTE:

This [browsing context](#)'s Documents' Window objects all implement the `WindowModal` interface.

9. Set all the flags in the new browsing context's [popup sandboxing flag set](#) that are set in the [active sandboxing flag set](#) of the [active document](#) of the [responsible browsing context](#) specified by the [incumbent settings object](#). The [responsible browsing context](#) specified by the [incumbent settings object](#) must be set as the new browsing context's [one permitted sandboxed navigator](#).
10. Let the [dialog arguments](#) of the new browsing context be set to the value of *argument*, or the *undefined* value if the argument was omitted.
11. Let the [dialog arguments' origin](#) be *incumbent origin*.
12. Let the [return value](#) of the new browsing context be the *undefined* value.
13. Let the [return value origin](#) be *incumbent origin*.
14. [Navigate](#) the new [browsing context](#) to the [absolute URL](#) that resulted from [parsing *url*](#) earlier, with [replacement enabled](#), and with the [responsible browsing context](#) specified by the [incumbent settings object](#) as the [source browsing context](#).
15. [Spin the event loop](#) until the new [browsing context](#) is closed. The user agent must allow the user to indicate that the [browsing context](#) is to be closed.
16. Reenable the user interface for all the browsing contexts in *the list of background browsing contexts*.
17. If the [auxiliary browsing context](#)'s [return value origin](#) at the time the browsing context was closed was the same as *incumbent origin*, then let *return value* be the [auxiliary browsing context](#)'s [return value](#) as it stood when the browsing context was closed.

Otherwise, let *return value* be undefined.
18. Return *return value*.

The Window objects of Documents hosted by [browsing contexts](#) created by the above algorithm must also implement the WindowModal interface.

NOTE:

When this happens, the members of the WindowModal interface, in JavaScript environments, appear to actually be part of the Window interface (e.g., they are on the same prototype chain as the `window.alert()` method).

```
[NoInterfaceObject]
interface WindowModal {
  readonly attribute any dialogArguments;
  attribute any returnValue;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

`window . dialogArguments`

Returns the *argument* argument that was passed to the `showModalDialog()` method.

`window . returnValue [= value ]`

Returns the current return value for the window.

Can be set, to change the value that will be returned by the `showModalDialog()` method.

Such browsing contexts have associated **dialog arguments**, which are stored along with the **dialog arguments' origin**. These values are set by the `showModalDialog()` method in the algorithm above, when the browsing context is created, based on the arguments provided to the method.

The **dialogArguments** IDL attribute, on getting, must check whether its browsing context's **active document's origin** is the **same origin-domain** as the **dialog arguments' origin**. If it is, then the browsing context's **dialog arguments** must be returned unchanged. Otherwise, the IDL attribute must return *undefined*.

These browsing contexts also have an associated **return value** and **return value origin**. As with the previous two values, these values are set by the `showModalDialog()` method in the algorithm above, when the browsing context is created.

The **returnValue** IDL attribute, on getting, must check whether its browsing context's **active document's origin** is the **same origin-domain** as the current **return value origin**. If it is, then the **browsing context's return value** must be returned unchanged. Otherwise, the IDL attribute must return *undefined*. On setting, the attribute must set the **return value** to the given new value, and the **return value origin** to the **browsing context's active document's origin**.

NOTE:

The `window.close()` method can be used to close the browsing context.

§ 7.6. System state and capabilities

§ 7.6.1. The Navigator object

The **navigator** attribute of the [Window](#) interface must return an instance of the Navigator interface, which represents the identity and state of the user agent (the client), and allows Web pages to register themselves as potential protocol and content handlers:

```
interface Navigator {
  // objects implementing this interface also implement the interfaces given below
};
Navigator implements NavigatorID;
Navigator implements NavigatorLanguage;
Navigator implements NavigatorOnLine;
Navigator implements NavigatorContentUtils;
Navigator implements NavigatorCookies;
Navigator implements NavigatorPlugins;
```

These interfaces are defined separately so that other specifications can re-use parts of the `Navigator` interface.

§ 7.6.1.1. Client identification

```
[NoInterfaceObject, Exposed=(Window, Worker)]
interface NavigatorID {
  [Exposed=Window] readonly attribute DOMString appCodeName; // constant "Mozilla"
  readonly attribute DOMString appName; // constant "Netscape"
  readonly attribute DOMString appVersion;
  readonly attribute DOMString platform;
  [Exposed=Window]readonly attribute DOMString product; // constant "Gecko"
  readonly attribute DOMString userAgent;
};
```

In certain cases, despite the best efforts of the entire industry, Web browsers have bugs and limitations that Web authors are forced to work around.

This section defines a collection of attributes that can be used to determine, from script, the kind of user agent in use, in order to work around these issues.

Client detection should always be limited to detecting known current versions; future versions and unknown versions should always be assumed to be fully compliant.

This definition is non-normative. Implementation requirements are given below this definition.

`window.navigator.appCodeName`

Returns the string "Mozilla".

`window.navigator.appName`

Returns the string "Netscape".

`window.navigator.appVersion`

Returns the version of the browser.

`window.navigator.platform`

Returns the name of the platform.

`window.navigator.product`

Returns the string "Gecko".

`window.navigator.taintEnabled()`

Returns either the string "20030107", or the string "20100101".

`window.navigator.userAgent`

Returns the complete User-Agent header.

`appCodeName`, of type [DOMString](#), readonly

Must return the string "Mozilla".

`appName`, of type [DOMString](#), readonly

Must return the string "Netscape".

`appVersion`, of type [DOMString](#), readonly

Must return either the string "4.0" or a string representing the version of the browser in detail, e.g., "1.0 (VMS; en-US) Mellblomenator/9000".

`platform`, of type [DOMString](#), readonly

Must return either the empty string or a string representing the platform on which the browser is executing, e.g., "MacIntel", "Win32", "FreeBSD i386", "WebTV OS".

`product`, of type [DOMString](#), readonly

Must return the string "Gecko".

taintEnabled()

Must return false.

***userAgent*, of type **DOMString**, readonly**

Must return the string used for the value of the "User-Agent" header in HTTP requests, or the empty string if no such header is ever sent.

⚠Warning! Any information in this API that varies from user to user can be used to profile the user. In fact, if enough such information is available, a user can actually be uniquely identified. For this reason, user agent implementors are strongly urged to include as little information in this API as possible.

7.6.1.2. Language preferences

```
[NoInterfaceObject, Exposed=(Window, Worker)]
interface NavigatorLanguage {
  readonly attribute DOMString? language;
  readonly attribute DOMString[] languages;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

window.navigator.language

Returns a language tag representing the user's preferred language.

window.navigator.languages

Returns an array of language tags representing the user's preferred languages, with the most preferred language first. The most preferred language is the one returned by `navigator.language`.

NOTE:

A `languagechange` event is fired at the `Window` or `WorkerGlobalScope` object when the user agent's understanding of what the user's preferred languages are changes.

***Language*, of type **DOMString**, readonly, nullable**

Must return a valid BCP 47 language tag representing either [a plausible language](#) or the user's most preferred language. [\[BCP47\]](#)

***Languages*, of type **DOMString[]**, readonly**

Must return a [read only](#) array of valid BCP 47 language tags representing either one or more [plausible languages](#), or the user's preferred languages, ordered by preference with the most preferred language first. The same object must be returned until the user agent needs to return different values, or values in a different order. [\[BCP47\]](#)

Whenever the user agent needs to make the `navigator.languages` attribute of a `Window` or `WorkerGlobalScope` object return a new set of language tags, the user agent must [queue a task](#) to [fire a simple event](#) named `languagechange` at the `Window` or `WorkerGlobalScope` object and wait until that task begins to be executed before actually returning a new value.

The [task source](#) for this [task](#) is the [DOM manipulation task source](#).

To determine a **plausible language**, the user agent should bear in mind the following:

- Any information in this API that varies from user to user can be used to profile or identify the user.
- If the user is not using a service that obfuscates the user's point of origin (e.g., the Tor anonymity network), then the value that is least likely to distinguish the user from other users with similar origins (e.g., from the same IP address block) is the language used by the majority of such users. [\[TOR\]](#)
- If the user is using an anonymizing service, then the value "en-US" is suggested; if all users of the service use that same

value, that reduces the possibility of distinguishing the users from each other.

To avoid introducing any more fingerprinting vectors, user agents should use the same list for the APIs defined in this function as for the HTTP Accept-Language header.

§ 7.6.1.3. Custom scheme handler: the `registerProtocolHandler()` method

```
[NoInterfaceObject]
interface NavigatorContentUtils {
  // content handler registration
  void registerProtocolHandler(DOMString scheme, DOMString url, DOMString title);
  void unregisterProtocolHandler(DOMString scheme, DOMString url);
};
```

The `registerProtocolHandler()` method allows Web sites to register themselves as possible handlers for particular schemes. For example, an online telephone messaging service could register itself as a handler of the `sms:` scheme, so that if the user clicks on such a link, he is given the opportunity to use that Web site. [\[RFC5724\]](#)

This definition is non-normative. Implementation requirements are given below this definition.

`window.navigator.registerProtocolHandler(scheme, url, title)`

Registers a handler for the given scheme, at the given URL, with the given title.

The string `"%s"` in the URL is used as a placeholder for where to put the URL of the content to be handled.

Throws a ["SecurityError" DOMException](#) if the user agent blocks the registration (this might happen if trying to register as a handler for `"http"`, for instance).

Throws a ["SyntaxError" DOMException](#) if the `"%s"` string is missing in the URL.

User agents may, within the constraints described in this section, do whatever they like when the method is called. A user agent could, for instance, prompt the user and offer the user the opportunity to add the site to a shortlist of handlers, or make the handler his default, or cancel the request. User agents could provide such a UI through modal UI or through a non-modal transient notification interface. user agents could also simply silently collect the information, providing it only when relevant to the user.

User agents should keep track of which sites have registered handlers (even if the user has declined such registrations) so that the user is not repeatedly prompted with the same request.

The arguments to the method have the following meanings and corresponding implementation requirements. The requirements that involve throwing exceptions must be processed in the order given below, stopping at the first exception thrown. (So the exceptions for the first argument take precedence over the exceptions for the second argument.)

`scheme` (`registerProtocolHandler()`)

A scheme, such as `"mailto"` or `"web+auth"`. The scheme must be compared in an [ASCII case-insensitive](#) manner by user agents for the purposes of comparing with the scheme part of URLs that they consider against the list of registered handlers.

The `scheme` value, if it contains a colon (as in `"mailto:"`), will never match anything, since schemes don't contain colons.

If the `registerProtocolHandler()` method is invoked with a scheme that is neither a [safelisted scheme](#) nor a scheme whose value starts with the substring `"web+"` and otherwise contains only [lowercase ASCII letters](#), and whose length is at least five characters (including the `"web+"` prefix), the user agent must throw a ["SyntaxError" DOMException](#).

The following schemes are the **safelisted schemes**:

- bitcoin
- geo
- im
- irc
- ircs
- magnet
- mailto

- mms
- news
- nntp
- openpgp4fpr
- sip
- sms
- smsto
- ssh
- tel
- urn
- webcal
- wtai
- xmpp

NOTE:

This list can be changed. If there are schemes that should be added, please send feedback.

NOTE:

This list excludes any schemes that could reasonably be expected to be supported inline, e.g., in an `<iframe>`, such as http or (more theoretically) gopher. If those were supported, they could potentially be used in man-in-the-middle attacks, by replacing pages that have frames with such content with content under the control of the protocol handler. If the user agent has native support for the schemes, this could further be used for cookie-theft attacks.

url

A string used to build the [URL](#) of the page that will handle the requests.

User agents must throw a "[SyntaxError](#)" [DOMException](#) if the *url* argument passed to one of these methods does not contain the exact literal string "%s".

User agents must throw a "[SyntaxError](#)" [DOMException](#) if [parsing](#) the *url* argument relative to the [API base URL](#) specified by the [entry settings object](#) is not successful.

NOTE:

The [resulting URL string](#) would by definition not be a [valid URL](#) as it would include the string "%s" which is not a valid component in a URL.

User agents must throw a "[SecurityError](#)" [DOMException](#) if the resulting [absolute URL](#) has an [origin](#) that differs from the [origin](#) specified by the [entry settings object](#).

NOTE:

This is forcibly the case if the %s placeholder is in the scheme, host, or port parts of the URL.

The [resulting URL string](#) is the **proto-URL**. It identifies the handler for the purposes of the methods described below.

When the user agent uses this handler, it must replace the first occurrence of the exact literal string "%s" in the *url* argument with an escaped version of the [absolute URL](#) of the content in question (as defined below), then [parse](#) the resulting URL, relative to the [API base URL](#) specified by the [entry settings object](#) at the time the `registerProtocolHandler()` method was invoked, and then [navigate](#) an appropriate [browsing context](#) to the resulting URL.

To get the escaped version of the [absolute URL](#) of the content in question, the user agent must replace every character in that [absolute URL](#) that is not a character in the URL [default encode set](#) with the result of [UTF-8 percent encoding](#) that character.

title

A descriptive title of the handler, which the user agent might use to remind the user what the site in question is.

This section does not define how the pages registered by these methods are used, beyond the requirements on how to process the *url* value (see above). To some extent, the [processing model for navigating across documents](#) defines some cases where these methods are relevant, but in general user agents may use this information wherever they would otherwise consider handing content to native plugins or helper applications.



In addition to the registration method, there is also a method for unregistering a handler.

This definition is non-normative. Implementation requirements are given below this definition.

`window.navigator.unregisterProtocolHandler(scheme, url)`

Unregisters the handler given by the arguments.



The **`unregisterProtocolHandler()`** method must unregister the handler described by the two arguments to the method, where the first argument gives the scheme and the second gives the string used to build the [URL](#) of the page that will handle the requests.

The first argument must be compared to the schemes for which custom protocol handlers are registered in an [ASCII case-insensitive](#) manner to find the relevant handlers.

The second argument must be preprocessed as described below, and if that is successful, must then be matched against the [proto-URLs](#) of the relevant handlers to find the described handler.



The second argument of the two methods described above must be preprocessed as follows:

1. If the string does not contain the substring "%s", abort these steps. There's no matching handler.
2. [Parse](#) the string relative to the [entry settings object](#). If this fails, then throw a **"SyntaxError" DOMException**.
3. If the resulting URL record's [origin](#) is not the [same origin](#) as the [origin](#) specified by the [entry settings object](#), throw a **"SecurityError" DOMException**.
4. Return the [resulting URL string](#) as the result of preprocessing the argument.

§ 7.6.1.3.1. SECURITY AND PRIVACY

These mechanisms can introduce a number of concerns, in particular privacy concerns.

Hijacking all Web usage. User agents should not allow schemes that are key to its normal operation, such as `http` or `https`, to be rerouted through third-party sites. This would allow a user's activities to be trivially tracked, and would allow user information, even in secure connections, to be collected.

Hijacking defaults. User agents are strongly urged to not automatically change any defaults, as this could lead the user to send data to remote hosts that the user is not expecting. New handlers registering themselves should never automatically cause those sites to be used.

Registration spamming. User agents should consider the possibility that a site will attempt to register a large number of handlers, possibly from multiple domains (e.g., by redirecting through a series of pages each on a different domain, and each registering a scheme handler — analogous practices abusing other Web browser features have been used by pornography Web sites for many years). User agents should gracefully handle such hostile attempts, protecting the user.

Misleading titles. User agents should not rely wholly on the *title* argument to the methods when presenting the registered handlers to the user, since sites could easily lie. For example, a site `hostile.example.net` could claim that it was registering the "Cuddly Bear Happy Scheme Handler". User agents should therefore use the handler's domain in any UI along with any title.

Hostile handler metadata. User agents should protect against typical attacks against strings embedded in their interface, for example ensuring that markup or escape characters in such strings are not executed, that null bytes are properly handled, that over-long strings do not cause crashes or buffer overruns, and so forth.

Leaking Intranet URLs. The mechanism described in this section can result in secret Intranet URLs being leaked, in the following manner:

1. The user registers a third-party content handler as the default handler for a scheme.
2. The user then browses his corporate Intranet site and accesses a link that uses that scheme.
3. The user agent contacts the third party and hands the third party the URL to the Intranet content.

No actual confidential file data is leaked in this manner, but the URLs themselves could contain confidential information. For example, the URL could be `cuddly://www.corp.example.com/upcoming-aquisitions/the-sample-company.egf`, which might tell the third party that Example Corporation is intending to merge with The Sample Company. Implementors might wish to consider allowing administrators to disable this feature for certain subdomains, or schemes.

Leaking credentials. User agents must never send username or password information in the URLs that are escaped and included sent to the handler sites. User agents may even avoid attempting to pass to Web-based handlers the URLs of resources that are known to require authentication to access, as such sites would be unable to access the resources in question without prompting the user for credentials themselves (a practice that would require the user to know whether to trust the third-party handler, a decision many users are unable to make or even understand).

Interface interference. User agents should be prepared to handle intentionally long arguments to the methods. For example, if the user interface exposed consists of an "accept" button and a "deny" button, with the "accept" binding containing the name of the handler, it's important that a long name not cause the "deny" button to be pushed off the screen.

Fingerprinting users. Since a site can detect if it has attempted to register a particular handler or not, whether or not the user responds, the mechanism can be used to store data. User agents are therefore strongly urged to treat registrations in the same manner as cookies: clearing cookies for a site should also clear all registrations for that site, and disabling cookies for a site should also disable registrations.

7.6.1.4. Cookies

```
[NoInterfaceObject]
interface NavigatorCookies {
  readonly attribute boolean cookieEnabled;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

window.navigator.cookieEnabled

Returns false if setting a cookie will be ignored, and true otherwise.

The **cookieEnabled** attribute must return true if the user agent attempts to handle cookies according to the cookie specification, and false if it ignores cookie change requests. [\[COOKIES\]](#)

7.6.1.5. Plugins

```
[NoInterfaceObject]
interface NavigatorPlugins {
  [SameObject] readonly attribute PluginArray plugins;
  [SameObject] readonly attribute MimeTypeArray mimeTypes;
  boolean javaEnabled();
};
```

```

interface PluginArray {
  void refresh(optional boolean reload = false);
  readonly attribute unsigned long length;
  getter Plugin? item(unsigned long index);
  getter Plugin? namedItem(DOMString name);
};

interface MimeTypeArray {
  readonly attribute unsigned long length;
  getter MimeType? item(unsigned long index);
  getter MimeType? namedItem(DOMString name);
};

interface Plugin {
  readonly attribute DOMString name;
  readonly attribute DOMString description;
  readonly attribute DOMString filename;
  readonly attribute unsigned long length;
  getter MimeType? item(unsigned long index);
  getter MimeType? namedItem(DOMString name);
};

interface MimeType {
  readonly attribute DOMString type;
  readonly attribute DOMString description;
  readonly attribute DOMString suffixes; // comma-separated
  readonly attribute Plugin enabledPlugin;
};

```

This definition is non-normative. Implementation requirements are given below this definition.

`window.navigator.plugins.refresh( [ refresh ] )`

Updates the lists of supported plugins and MIME types for this page, and reloads the page if the lists have changed.

`window.navigator.plugins.length`

Returns the number of plugins, represented by Plugin objects, that the user agent reports.

`plugin = window.navigator.plugins.item(index)`

`window.navigator.plugins[index]`

Returns the specified Plugin object.

`plugin = window.navigator.plugins.item(name)`

`window.navigator.plugins[name]`

Returns the Plugin object for the plugin with the given name.

`window.navigator.mimeTypes.length`

Returns the number of MIME types, represented by MimeType objects, supported by the plugins that the user agent reports.

`mimeType = window.navigator.mimeTypes.item(index)`

`window.navigator.mimeTypes[index]`

Returns the specified MimeType object.

`mimeType = window.navigator.mimeTypes.item(name)`

`window.navigator.mimeTypes[name]`

Returns the MimeType object for the given MIME type.

`plugin.name`

Returns the plugin's name.

`plugin .description`

Returns the plugin's description.

`plugin .filename`

Returns the plugin library's filename, if applicable on the current platform.

`plugin .length`

Returns the number of MIME types, represented by `MimeType` objects, supported by the plugin.

`MimeType = plugin .item(index)`**`plugin [index]`**

Returns the specified `MimeType` object.

`MimeType = plugin .item(name)`**`plugin [name]`**

Returns the `MimeType` object for the given MIME type.

`MimeType .type`

Returns the MIME type.

`MimeType .description`

Returns the MIME type's description.

`MimeType .suffixes`

Returns the MIME type's typical file extensions, in a comma-separated list.

`MimeType .enabledPlugin`

Returns the `Plugin` object that implements this MIME type.

`window .navigator .javaEnabled()`

Returns true if there's a plugin that supports the MIME type "application/x-java-vm".

The **plugins** attribute must return a `PluginArray` object.

The **mimeTypes** attribute must return a `MimeTypeArray` object.



A `PluginArray` object represents none, some, or all of the [plugins](#) supported by the user agent, each of which is represented by a `Plugin` object. Each of these `Plugin` objects may be **hidden plugins**. A [hidden plugin](#) can't be enumerated, but can still be inspected by using its name.

NOTE:

The fewer [plugins](#) are represented by the `PluginArray` object, and of those, the more that are [hidden](#), the more the user's privacy will be protected. Each exposed plugin increases the number of bits that can be derived for fingerprinting. Hiding a plugin helps, but unless it is an extremely rare plugin, it is likely that a site attempting to derive the list of plugins can still determine whether the plugin is supported or not by probing for it by name (the names of popular plugins are widely known). Therefore not exposing a plugin at all is preferred. Unfortunately, many legacy sites use this feature to determine, for example, which plugin to use to play video. Not exposing any plugins at all might therefore not be entirely plausible.

The `PluginArray` objects created by a user agent must not be [live](#). The set of plugins represented by the objects must not change once an object is created, except when it is updated by the `refresh()` method.

Each [plugin](#) represented by a `PluginArray` can support a number of [MIME types](#). For each such [plugin](#), the user agent must pick one or more of these [MIME types](#) to be those that are **explicitly supported**.

NOTE:

The [explicitly supported MIME types](#) of a [plugin](#) are those that are exposed through the `Plugin` and `MimeTypeArray` interfaces. As with [plugins](#) themselves, any variation between users regarding what is exposed allows sites to fingerprint users. User agents are therefore encouraged to expose the same [MIME types](#) for all users of a [plugin](#), regardless of the actual types supported... at least, within the constraints imposed by compatibility with legacy content.

The [supported property indices](#) of a `PluginArray` object are the numbers from zero to the number of non-[hidden plugins](#) represented by the object, if any.

The **length** attribute must return the number of non-[hidden plugins](#) represented by the object.

The **item()** method of a `PluginArray` object must return null if the argument is not one of the object's [supported property indices](#), and otherwise must return the result of running the following steps, using the method's argument as *index*:

1. Let *list* be the `Plugin` objects representing the non-[hidden plugins](#) represented by the `PluginArray` object.
2. Sort *list* alphabetically by the name of each `Plugin`.
3. Return the *index*th entry in *list*.

NOTE:

It is important for privacy that the order of plugins not leak additional information, e.g., the order in which plugins were installed.

The [supported property names](#) of a `PluginArray` object are the values of the name attributes of all the `Plugin` objects represented by the `PluginArray` object. The properties exposed in this way must be [unenumerable](#).

The **namedItem()** method of a `PluginArray` object must return null if the argument is not one of the object's [supported property names](#), and otherwise must return the `Plugin` object, of those represented by the `PluginArray` object, that has a name equal to the method's argument.

The **refresh()** method of the `PluginArray` object of a `Navigator` object, when invoked, must check to see if any [plugins](#) have been installed or reconfigured since the user agent created the `PluginArray` object. If so, and the method's argument is true, then the user agent must act as if the `location.reload()` method was called instead. Otherwise, the user agent must update the `PluginArray` object and `MimeTypeArray` object created for attributes of that `Navigator` object, and the `Plugin` and `MimeType` objects created for those `PluginArray` and `MimeTypeArray` objects, using the same `Plugin` objects for cases where the name is the same, and the same `MimeType` objects for cases where the type is the same, and creating new objects for cases where there were no matching objects immediately prior to the `refresh()` call. Old `Plugin` and `MimeType` objects must continue to return the same values that they had prior to the update, though naturally now the data is stale and may appear inconsistent (for example, an old `MimeType` entry might list as its `enabledPlugin` a `Plugin` object that no longer lists that `MimeType` as a supported `MimeType`).



A `MimeTypeArray` object represents the [MIME types explicitly supported](#) by [plugins](#) supported by the user agent, each of which is represented by a `MimeType` object.

The `MimeTypeArray` objects created by a user agent must not be [live](#). The set of MIME types represented by the objects must not change once an object is created, except when it is updated by the `PluginArray` object's `refresh()` method.

The [supported property indices](#) of a `MimeTypeArray` object are the numbers from zero to the number of [MIME types explicitly supported](#) by non-[hidden plugins](#) represented by the corresponding `PluginArray` object, if any.

The **length** attribute must return the number of [MIME types explicitly supported](#) by non-[hidden plugins](#) represented by the corresponding `PluginArray` object, if any.

The **item()** method of a `MimeTypeArray` object must return null if the argument is not one of the object's [supported property indices](#), and otherwise must return the result of running the following steps, using the method's argument as *index*:

1. Let *list* be the `MimeType` objects representing the [MIME types explicitly supported](#) by non-[hidden plugins](#) represented by the corresponding `PluginArray` object, if any.

2. Sort *list* alphabetically by the type of each `MimeType`.

3. Return the *index*th entry in *list*.

NOTE:

It is important [for privacy](#) that the order of MIME types not leak additional information, e.g., the order in which plugins were installed.

The [supported property names](#) of a `MimeTypeArray` object are the values of the `type` attributes of all the `MimeType` objects represented by the `MimeTypeArray` object. The properties exposed in this way must be [unenumerable](#).

The `namedItem()` method of a `MimeTypeArray` object must return null if the argument is not one of the object's [supported property names](#), and otherwise must return the `MimeType` object that has a type equal to the method's argument.



A `Plugin` object represents a [plugin](#). It has several attributes to provide details about the plugin, and can be enumerated to obtain the list of [MIME types](#) that it [explicitly supports](#).

The `Plugin` objects created by a user agent must not be [live](#). The set of MIME types represented by the objects, and the values of the objects' attributes, must not change once an object is created, except when updated by the `PluginArray` object's `refresh()` method.

The **reported MIME types** for a `Plugin` object are the [MIME types explicitly supported](#) by the corresponding [plugin](#) when this object was last created or updated by `PluginArray.refresh()`, whichever happened most recently.

The [supported property indices](#) of a `Plugin` object are the numbers from zero to the number of [reported MIME types](#).

The **length** attribute must return the number of [reported MIME types](#).

The `item()` method of a `Plugin` object must return null if the argument is not one of the object's [supported property indices](#), and otherwise must return the result of running the following steps, using the method's argument as *index*:

1. Let *list* be the `MimeType` objects representing the [reported MIME types](#).
2. Sort *list* alphabetically by the type of each `MimeType`.
3. Return the *index*th entry in *list*.

NOTE:

It is important [for privacy](#) that the order of MIME types not leak additional information, e.g., the order in which plugins were installed.

The [supported property names](#) of a `Plugin` object are the values of the `type` attributes of the `MimeType` objects representing the [reported MIME types](#). The properties exposed in this way must be [unenumerable](#).

The `namedItem()` method of a `Plugin` object must return null if the argument is not one of the object's [supported property names](#), and otherwise must return the `MimeType` object that has a type equal to the method's argument.

The **name** attribute must return the [plugin](#)'s name.

The **description** and **filename** attributes must return user-agent-defined (or, in all likelihood, [plugin](#)-defined) strings. In each case, the same string must be returned each time, except that the strings returned may change when the `PluginArray.refresh()` method updates the object.

⚠Warning! If the values returned by the `description` or `filename` attributes vary between versions of a [plugin](#), they can be used both as a fingerprinting vector and, even more importantly, as a trivial way to determine what security vulnerabilities a [plugin](#) (and thus a browser) may have. It is thus highly recommended that the `description` attribute just return the same value as the `name` attribute, and that the `filename` attribute return the empty string.



A `MimeType` object represents a [MIME type](#) that is, or was, [explicitly supported](#) by a [plugin](#).

The `MimeType` objects created by a user agent must not be [live](#). The values of the objects' attributes must not change once an object is created, except when updated by the `PluginArray` object's `refresh()` method.

The `type` attribute must return the [valid MIME type with no parameters](#) describing the [MIME type](#).

The `description` and `suffixes` attributes must return user-agent-defined (or, in all likelihood, [plugin](#)-defined) strings. In each case, the same string must be returned each time, except that the strings returned may change when the `PluginArray.refresh()` method updates the object.

⚠Warning! If the values returned by the `description` or `suffixes` attributes vary between versions of a [plugin](#), they can be used both as a fingerprinting vector and, even more importantly, as a trivial way to determine what security vulnerabilities a [plugin](#) (and thus a browser) may have. It is thus highly recommended that the `description` attribute just return the same value as the `type` attribute, and that the `suffixes` attribute return the empty string.

NOTE:

Commas in the `suffixes` attribute are interpreted as separating subsequent filename extensions, as in "htm,html".

The `enabledPlugin` attribute must return the `Plugin` object that represents the [plugin](#) that [explicitly supported](#) the [MIME type](#) that this `MimeType` object represents when this object was last created or updated by `PluginArray.refresh()`, which ever happened most recently.



The `javaEnabled()` attribute must return true if the user agent supports a [plugin](#) that supports the [MIME type](#) "application/x-java-vm"; otherwise it must return false.

§ 7.7. Images

```
[Exposed=(Window, Worker)]
interface ImageBitmap {
  readonly attribute unsigned long width;
  readonly attribute unsigned long height;
};

typedef (HTMLImageElement or
HTMLVideoElement or
HTMLCanvasElement or
Blob or
ImageData or
CanvasRenderingContext2D or
ImageBitmap) ImageBitmapSource;

[NoInterfaceObject, Exposed=(Window, Worker)]
interface ImageBitmapFactories {
  Promise<ImageBitmap> createImageBitmap(ImageBitmapSource image);
  Promise<ImageBitmap> createImageBitmap(ImageBitmapSource image, long sx, long sy, long sw,
long sh);
};
Window implements ImageBitmapFactories;
WorkerGlobalScope implements ImageBitmapFactories;
```

An `ImageBitmap` object represents a bitmap image that can be painted to a canvas without undue latency.

NOTE:

The exact judgement of what is undue latency of this is left up to the implementer, but in general if making use of the bitmap requires network I/O, or even local disk I/O, then the latency is probably undue; whereas if it only requires a blocking read from a GPU or system RAM, the latency is probably acceptable.

This definition is non-normative. Implementation requirements are given below this definition.

`promise = Window.createImageBitmap(image [, sx, sy, sw, sh ] )`

Takes *image*, which can be an [](#) element, [<video>](#), or [<canvas>](#) element, a Blob object, an ImageData object, a CanvasRenderingContext2D object, or another ImageBitmap object, and returns a promise that is resolved when a new ImageBitmap is created.

If no ImageBitmap object can be constructed, for example because the provided *image* data is not actually an image, then the promise is rejected instead.

If *sx*, *sy*, *sw*, and *sh* arguments are provided, the source image is cropped to the given pixels, with any pixels missing in the original replaced by transparent black. These coordinates are in the source image's pixel coordinate space, *not* in CSS pixels.

Rejects the promise with an `InvalidStateError` exception if the source image is not in a valid state (e.g., an [](#) element that hasn't finished loading, or a CanvasRenderingContext2D object whose bitmap data has zero length along one or both dimensions, or an ImageData object whose data attribute has been neutered). Rejects the promise with a `"SyntaxError" DOMException` if the script is not allowed to access the image data of the source image (e.g., a video that is [CORS-cross-origin](#), or a canvas being drawn on by a script in a worker from another [origin](#)).

`imageBitmap . width`

Returns the [intrinsic width](#) of the image, in CSS pixels.

`imageBitmap . height`

Returns the [intrinsic height](#) of the image, in CSS pixels.

An ImageBitmap object always has associated bitmap data, with a width and a height. However, it is possible for this data to be corrupted. If an ImageBitmap object's media data can be decoded without errors, it is said to be [fully decodable](#).

An ImageBitmap object's bitmap has an [origin-clean](#) flag, which indicates whether the bitmap is tainted by content from a different [origin](#). The flag is initially set to true and may be changed to false by the steps of `createImageBitmap()`.

An ImageBitmap object can be obtained from a variety of different objects, using the `createImageBitmap()` method. When invoked, the method must act as follows:

If *image* is an [](#) element

1. If either the *sw* or *sh* arguments are specified but zero, return a promise rejected with an `IndexSizeError` exception and abort these steps.
2. If the [](#) element is not [completely available](#), then return a promise rejected with an `InvalidStateError` exception and abort these steps.
3. If the [](#) element's media data is not a bitmap (e.g., it's a vector graphic), then return a promise rejected with an `InvalidStateError` exception and abort these steps.
4. Create a new ImageBitmap object.
5. Let the ImageBitmap object's bitmap data be a copy of the `img` element's media data, [cropped to the source rectangle](#). If this is an animated image, the ImageBitmap object's bitmap data must only be taken from the default image of the animation (the one that the format defines is to be used when animation is not supported or is disabled), or, if there is no such image, the first frame of the animation.
6. If the [origin](#) of the [](#) element's image is not the [same origin](#) as the [origin](#) specified by the [entry settings object](#), then set the [origin-clean](#) flag of the ImageBitmap object's bitmap to false.
7. Return a new promise, but continue running these steps [in parallel](#).

8. Resolve the promise with the new `ImageBitmap` object as the value.

If *image* is a `<video>` element

1. If either the `sw` or `sh` arguments are specified but zero, return a promise rejected with an `IndexSizeError` exception and abort these steps.
2. If the `<video>` element's `networkState` attribute is `NETWORK_EMPTY`, then return a promise rejected with an `InvalidStateError` exception and abort these steps.
3. If the `<video>` element's `readyState` attribute is either `HAVE_NOTHING` or `HAVE_METADATA`, then return a promise rejected with an `InvalidStateError` exception and abort these steps.
4. Create a new `ImageBitmap` object.
5. Let the `ImageBitmap` object's bitmap data be a copy of the frame at the [current playback position](#), at the [media resource's intrinsic width](#) and [intrinsic height](#) (i.e., after any aspect-ratio correction has been applied), [cropped to the source rectangle](#).
6. If the [origin](#) of the `<video>` element's image is not the [same origin](#) as the [origin](#) specified by the [entry settings object](#), then set the [origin-clean](#) flag of the `ImageBitmap` object's bitmap to false.
7. Return a new promise, but continue running these steps [in parallel](#).
8. Resolve the promise with the new `ImageBitmap` object as the value.

If *image* is a `<canvas>` element

1. If either the `sw` or `sh` arguments are specified but zero, return a promise rejected with an `IndexSizeError` exception and abort these steps.
2. If the `<canvas>` element's bitmap has either a horizontal dimension or a vertical dimension equal to zero, then return a promise rejected with an `InvalidStateError` exception and abort these steps.
3. Create a new `ImageBitmap` object.
4. Let the `ImageBitmap` object's bitmap data be a copy of the `<canvas>` element's bitmap data, [cropped to the source rectangle](#).
5. Set the [origin](#) of the `ImageBitmap` object's bitmap to the same value as the [origin-clean](#) flag of the `<canvas>` element's bitmap.
6. Return a new promise, but continue running these steps [in parallel](#).
7. Resolve the promise with the new `ImageBitmap` object as the value.

If *image* is a `Blob` object

1. If either the `sw` or `sh` arguments are specified but zero, return a promise rejected with an `IndexSizeError` exception and abort these steps.

2. **⚠Warning!** There are no known native implementations of the File API [closed](#) method. Therefore this feature cannot be relied upon.

If *image* is [closed](#), then return a promise rejected with an `InvalidStateError` exception and abort these steps.

3. Return a new promise, but continue running these steps [in parallel](#).
4. Read the `Blob` object's data. If an [error occurs during reading of the object](#), then reject the promise with null, and abort these steps.
5. Apply the [image sniffing rules](#) to determine the file format of the image data, with MIME type of the `Blob` (as given by the `Blob` object's type attribute) giving the official type.
6. If the image data is not in a supported file format (e.g., it's not actually an image at all), or if the image data is cor-

rupted in some fatal way such that the image dimensions cannot be obtained, then reject the promise with null, and abort these steps.

7. Create a new `ImageBitmap` object.
8. Let the `ImageBitmap` object's bitmap data be the image data read from the `Blob` object, [cropped to the source rectangle](#). If this is an animated image, the `ImageBitmap` object's bitmap data must only be taken from the default image of the animation (the one that the format defines is to be used when animation is not supported or is disabled), or, if there is no such image, the first frame of the animation.
9. Resolve the promise with the new `ImageBitmap` object as the value.

If `image` is an `ImageData` object

1. If either the `sw` or `sh` arguments are specified but zero, return a promise rejected with an `IndexSizeError` exception and abort these steps.
2. If the `image` object's `data` attribute has been neutered, return a promise rejected with an `InvalidStateError` exception and abort these steps.
3. Create a new `ImageBitmap` object.
4. Let the `ImageBitmap` object's bitmap data be the image data given by the `ImageData` object, [cropped to the source rectangle](#).
5. Return a new promise, but continue running these steps [in parallel](#).
6. Resolve the promise with the new `ImageBitmap` object as the value.

If `image` is a `CanvasRenderingContext2D` object

1. If either the `sw` or `sh` arguments are specified but zero, return a promise rejected with an `IndexSizeError` exception and abort these steps.
2. If the `CanvasRenderingContext2D` object's scratch bitmap has either a horizontal dimension or a vertical dimension equal to zero, then return a promise rejected with an `InvalidStateError` exception and abort these steps.
3. Create a new `ImageBitmap` object.
4. Let the `ImageBitmap` object's bitmap data be a copy of the `CanvasRenderingContext2D` object's scratch bitmap, [cropped to the source rectangle](#).
5. Set the [origin-clean](#) flag of the `ImageBitmap` object's bitmap to the same value as the [origin-clean](#) flag of the `CanvasRenderingContext2D` object's scratch bitmap.
6. Return a new promise, but continue running these steps [in parallel](#).
7. Resolve the promise with the new `ImageBitmap` object as the value.

If `image` is an `ImageBitmap` object

1. If either the `sw` or `sh` arguments are specified but zero, return a promise rejected with an `IndexSizeError` exception and abort these steps.
2. Create a new `ImageBitmap` object.
3. Let the `ImageBitmap` object's bitmap data be a copy of the `image` argument's bitmap data, [cropped to the source rectangle](#).
4. Set the [origin-clean](#) flag of the `ImageBitmap` object's bitmap to the same value as the [origin-clean](#) flag of the bitmap of the `image` argument.
5. Return a new promise, but continue running these steps [in parallel](#).
6. Resolve the promise with the new `ImageBitmap` object as the value.

When the steps above require that the user agent **crop bitmap data to the source rectangle**, the user agent must run the fol-

lowing steps:

1. Let *input* be the image data being cropped.
2. If the *sx*, *sy*, *sw*, and *sh* arguments are omitted, return *input*.
3. Place *input* on an infinite transparent black grid plane, positioned so that its top left corner is at the origin of the plane, with the *x*-coordinate increasing to the right, and the *y*-coordinate increasing down, and with each pixel in the *input* image data occupying a cell on the plane's grid.
4. Let *output* be the rectangle on the plane denoted by the rectangle whose corners are the four points (*sx*, *sy*), (*sx*+*sw*, *sy*), (*sx*+*sw*, *sy*+*sh*), (*sx*, *sy*+*sh*).

NOTE:

If either *sw* or *sh* are negative, then the top-left corner of this rectangle will be to the left or above the (*sx*, *sy*) point. If any of the pixels on this rectangle are outside the area where the *input* bitmap was placed, then they will be transparent black in *output*.

5. Return *output*.

The **width** attribute must return the ImageBitmap object's width, in CSS pixels.

The **height** attribute must return the ImageBitmap object's height, in CSS pixels.

EXAMPLE 634

Using this API, a sprite sheet can be precut and prepared:

```
var sprites = {};
function loadMySprites() {
  var image = new Image();
  image.src = 'mysprites.png';
  var resolver;
  var promise = new Promise(function (arg) { resolver = arg });
  image.onload = function () {
    resolver(Promise.all(
      createImageBitmap(image, 0, 0, 40, 40).then(function (image) { sprites.woman = image }),
      createImageBitmap(image, 40, 0, 40, 40).then(function (image) { sprites.man = image }),
      createImageBitmap(image, 80, 0, 40, 40).then(function (image) { sprites.tree = image }),
      createImageBitmap(image, 0, 40, 40, 40).then(function (image) { sprites.hut = image }),
      createImageBitmap(image, 40, 40, 40, 40).then(function (image) { sprites.apple = image }),
      createImageBitmap(image, 80, 40, 40, 40).then(function (image) { sprites.snake = image })
    ));
  };
  return promise;
}

function runDemo() {
  var canvas = document.querySelector('canvas#demo');
  var context = canvas.getContext('2d');
  context.drawImage(sprites.tree, 30, 10);
  context.drawImage(sprites.snake, 70, 10);
}

loadMySprites().then(runDemo);
```

§ 7.8. Animation Frames

Each [Document](#) associated with a [top-level browsing context](#) has a **list of animation frame callbacks**, which must be initially empty, and an **animation frame callback identifier**, which is a number which must initially be zero.

When the **requestAnimationFrame()** method is called, the user agent must run the following steps:

1. Let *document* be the [Window](#) object's [Document](#) object
2. Increment *document*'s [animation frame callback identifier](#) by one.
3. Append the method's argument to *document*'s [list of animation frame callbacks](#), associated with *document*'s [animation frame callback identifier](#)'s current value
4. Return *document*'s [animation frame callback identifier](#)'s current value

When the **cancelAnimationFrame()** method is called, the user agent must run the following steps:

1. Let *document* be the [Window](#) object's [Document](#) object
2. Find the entry in *document*'s [list of animation frame callbacks](#) that is associated with the value given by the method's argument *handle*
3. If there is such an entry, remove it from *document*'s [list of animation frame callbacks](#)

When the user agent is to **run the animation frame callbacks** for a [Document](#) *document* with a timestamp *now*, it must run the following steps:

1. If the value returned by the *document* object's [hidden](#) attribute is true, abort these steps. [\[PAGE-VISIBILITY\]](#)
2. Let *callbacks* be a list of the entries in *document*'s [list of animation frame callbacks](#), in the order in which they were added to the list.
3. Set *document*'s [list of animation frame callbacks](#) to the empty list.
4. For each entry in *callbacks*, in order: [invoke the Web IDL callback function](#), passing *now* as the only argument, and if an exception is thrown, [report the exception](#). [\[WEBIDL\]](#)

§ 8. The HTML syntax

NOTE:

This section only describes the rules for resources labeled with an [HTML MIME type](#). Rules for XML resources are discussed in the section below entitled "[the XHTML syntax](#)".

§ 8.1. Writing HTML documents

This section only applies to documents, authoring tools, and markup generators. In particular, it does not apply to conformance checkers; conformance checkers must use the requirements given in the next section ("parsing HTML documents").

Documents must consist of the following parts, in the given order:

1. Optionally, a single U+FEFF BYTE ORDER MARK (BOM) character.
2. Any number of [comments](#) and [space characters](#).
3. A [DOCTYPE](#).
4. Any number of [comments](#) and [space characters](#).
5. The root element, in the form of an [<html>](#) element.
6. Any number of [comments](#) and [space characters](#).

The various types of content mentioned above are described in the next few sections.

In addition, there are some restrictions on how [character encoding declarations](#) are to be serialized, as discussed in the section on that topic.

Space characters before the root `<html>` element, and space characters at the start of the `<html>` element and before the `<head>` element, will be dropped when the document is parsed; space characters after the root `<html>` element will be parsed as if they were at the end of the `<body>` element. Thus, space characters around the root element do not round-trip.

It is suggested that newlines be inserted after the DOCTYPE, after any comments that are before the root element, after the `<html>` element's start tag (if it is not [omitted](#)), and after any comments that are inside the `<html>` element but before the `<head>` element.

Many strings in the HTML syntax (e.g., the names of elements and their attributes) are case-insensitive, but only for [uppercase ASCII letters](#) and [lowercase ASCII letters](#). For convenience, in this section this is just referred to as "case-insensitive".

§ 8.1.1. The DOCTYPE

A **DOCTYPE** is a required preamble.

NOTE:

DOCTYPES are required for legacy reasons. When omitted, browsers tend to use a different rendering mode that is incompatible with some specifications. Including the DOCTYPE in a document ensures that the browser makes a best-effort attempt at following the relevant specifications.

A DOCTYPE must consist of the following components, in this order:

1. A string that is an [ASCII case-insensitive](#) match for the string "`<!DOCTYPE`".
2. One or more [space characters](#).
3. A string that is an [ASCII case-insensitive](#) match for the string "`html`".
4. Optionally, a [DOCTYPE legacy string](#) or an [obsolete permitted DOCTYPE string](#) (defined below).
5. Zero or more [space characters](#).
6. A U+003E GREATER-THAN SIGN character (`>`).

NOTE:

In other words, `<!DOCTYPE html>`, case-insensitively.



For the purposes of HTML generators that cannot output HTML markup with the short DOCTYPE "`<!DOCTYPE html>`", a **DOCTYPE legacy string** may be inserted into the DOCTYPE (in the position defined above). This string must consist of:

1. One or more [space characters](#).
2. A string that is an [ASCII case-insensitive](#) match for the string "`SYSTEM`".
3. One or more [space characters](#).
4. A U+0022 QUOTATION MARK or U+0027 APOSTROPHE character (the *quote mark*).
5. The literal string "`about:legacy-compat`".
6. A matching U+0022 QUOTATION MARK or U+0027 APOSTROPHE character (i.e., the same character as in the earlier step labeled *quote mark*).

NOTE:

In other words, `<!DOCTYPE html SYSTEM "about:legacy-compat">` or `<!DOCTYPE html SYSTEM 'about:legacy-compat'>`, case-insensitively except for the part in single or double quotes.

The [DOCTYPE legacy string](#) should not be used unless the document is generated from a system that cannot output the shorter string.



To help authors transition from HTML 4.01 and XHTML 1.1, an **obsolete permitted DOCTYPE string** can be inserted into the DOCTYPE (in the position defined above). This string must consist of:

1. One or more [space characters](#).
2. A string that is an [ASCII case-insensitive](#) match for the string "`PUBLIC`".
3. One or more [space characters](#).
4. A U+0022 QUOTATION MARK or U+0027 APOSTROPHE character (the *first quote mark*).
5. The string from one of the cells in the first column of the table below. The row to which this cell belongs is the *selected*

- row*.
6. A matching U+0022 QUOTATION MARK or U+0027 APOSTROPHE character (i.e., the same character as in the earlier step labeled *first quote mark*).
 7. If a system identifier is used,
 1. One or more [space characters](#).
 2. A U+0022 QUOTATION MARK or U+0027 APOSTROPHE character (the *third quote mark*).
 3. The string from the cell in the second column of the *selected row*.
 4. A matching U+0022 QUOTATION MARK or U+0027 APOSTROPHE character (i.e., the same character as in the earlier step labeled *third quote mark*).

Allowed values for public and system identifiers in an [obsolete permitted DOCTYPE string](#).

Public identifier	System identifier	System identifier optional?
-//W3C//DTD HTML 4.0//EN	http://www.w3.org/TR/REC-html40/strict.dtd	Yes
-//W3C//DTD HTML 4.01//EN	http://www.w3.org/TR/html4/strict.dtd	Yes
-//W3C//DTD XHTML 1.0 Strict//EN	http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd	No
-//W3C//DTD XHTML 1.1//EN	http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd	No

A [DOCTYPE](#) containing an [obsolete permitted DOCTYPE string](#) is an **obsolete permitted DOCTYPE**. Authors should not use [obsolete permitted DOCTYPEs](#), as they are unnecessarily long.

§ 8.1.2. Elements

There are five different **kinds of elements**: [void elements](#), [raw text elements](#), [escapable raw text elements](#), [foreign elements](#), and [normal elements](#).

Void elements

[<area>](#), [<base>](#), [
](#), [<col>](#), [<embed>](#), [<hr>](#), [](#), [<input>](#), [<keygen>](#), [<link>](#), [<menuitem>](#), [<meta>](#), [<param>](#), [<source>](#), [<track>](#), [<wbr>](#)

Raw text elements

[<script>](#), [<style>](#)

Escapable raw text elements

[<textarea>](#), [<title>](#)

Foreign elements

Elements from the [MathML namespace](#) and the [SVG namespace](#).

Normal elements

All other allowed [html elements](#) are normal elements.

Tags are used to delimit the start and end of elements in the markup. [Raw text](#), [escapable raw text](#), and [normal elements](#) have a [start tag](#) to indicate where they begin, and an [end tag](#) to indicate where they end. The start and end tags of certain [normal elements](#) can be [omitted](#), as described below in the section on [optional tags](#). Those that cannot be omitted must not be omitted. [Void elements](#) only have a start tag; end tags must not be specified for [void elements](#). [Foreign elements](#) must either have a start tag and an end tag, or a start tag that is marked as self-closing, in which case they must not have an end tag.

The [contents](#) of the element must be placed between just after the start tag (which [might be implied, in certain cases](#)) and just before the end tag (which again, [might be implied, in certain cases](#)). The exact allowed contents of each individual element depend on the [content model](#) of that element, as described earlier in this specification. Elements must not contain content that their content model disallows. In addition to the restrictions placed on the contents by those content models, however, the five types of elements have additional *syntactic* requirements.

[Void elements](#) can't have any contents (since there's no end tag, no content can be put between the start tag and the end tag).

[Raw text elements](#) can have [text](#), though it has [restrictions](#) described below.

[Escapable raw text elements](#) can have [text](#) and [character references](#), but the text must not contain an [ambiguous ampersand](#). There are also [further restrictions](#) described below.

[Foreign elements](#) whose start tag is marked as self-closing can't have any contents (since, again, as there's no end tag, no

content can be put between the start tag and the end tag). [Foreign elements](#) whose start tag is *not* marked as self-closing can have [text](#), [character references](#), [CDATA sections](#), other [elements](#), and [comments](#), but the text must not contain the character U+003C LESS-THAN SIGN (<) or an [ambiguous ampersand](#).

The HTML syntax does not support namespace declarations, even in [foreign elements](#).

For instance, consider the following HTML fragment:

```
<p>
<svg>
  <metadata>
    <!-- this is invalid -->
    <cdr:license xmlns:cdr="https://www.example.com/cdr/metadata" name="MIT"/>
  </metadata>
</svg>
</p>
```

The innermost element, `cdr:license`, is actually in the SVG namespace, as the `xmlns:cdr` attribute has no effect (unlike in XML). In fact, as the comment in the fragment above says, the fragment is actually non-conforming. This is because the SVG specification does not define any elements called `"cdr:license"` in the SVG namespace.

[Normal elements](#) can have [text](#), [character references](#), other [elements](#), and [comments](#), but the text must not contain the character U+003C LESS-THAN SIGN (<) or an [ambiguous ampersand](#). Some [normal elements](#) also have [yet more restrictions](#) on what content they are allowed to hold, beyond the restrictions imposed by the content model and those described in this paragraph. Those restrictions are described below.

Tags contain a **tag name**, giving the element's name. HTML elements all have names that only use [alphanumeric ASCII characters](#). In the HTML syntax, tag names, even those for [foreign elements](#), may be written with any mix of lower- and uppercase letters that, when converted to all-lowercase, matches the element's tag name; tag names are case-insensitive.

8.1.2.1. Start tags

Start tags must have the following format:

1. The first character of a start tag must be a U+003C LESS-THAN SIGN character (<).
2. The next few characters of a start tag must be the element's [tag name](#).
3. If there are to be any attributes in the next step, there must first be one or more [space characters](#).
4. Then, the start tag may have a number of attributes, the [syntax for which](#) is described below. Attributes must be separated from each other by one or more [space characters](#).
5. After the attributes, or after the [tag name](#) if there are no attributes, there may be one or more [space characters](#). (Some attributes are required to be followed by a space. See [§8.1.2.3 Attributes](#) below.)
6. Then, if the element is one of the [void elements](#), or if the element is a [foreign element](#), then there may be a single U+002F SOLIDUS character (/). This character has no effect on [void elements](#), but on [foreign elements](#) it marks the start tag as self-closing.
7. Finally, start tags must be closed by a U+003E GREATER-THAN SIGN character (>).

8.1.2.2. End tags

End tags must have the following format:

1. The first character of an end tag must be a U+003C LESS-THAN SIGN character (<).
2. The second character of an end tag must be a U+002F SOLIDUS character (/).
3. The next few characters of an end tag must be the element's [tag name](#).

4. After the tag name, there may be one or more [space characters](#).
5. Finally, end tags must be closed by a U+003E GREATER-THAN SIGN character (>).

8.1.2.3. Attributes

Attributes for an element are expressed inside the element's start tag.

Attributes have a name and a value. **Attribute names** must consist of one or more characters other than the [space characters](#), U+0000 NULL, U+0022 QUOTATION MARK ("), U+0027 APOSTROPHE ('), U+003E GREATER-THAN SIGN (>), U+002F SOLIDUS (/), and U+003D EQUALS SIGN (=) characters, the [control characters](#), and any characters that are not defined by Unicode. In the HTML syntax, attribute names, even those for [foreign elements](#), may be written with any mix of lower- and uppercase letters that are an [ASCII case-insensitive](#) match for the attribute's name.

Attribute values are a mixture of [text](#) and [character references](#), except with the additional restriction that the text cannot contain an [ambiguous ampersand](#).

Attributes can be specified in four different ways:

Empty attribute syntax

Just the [attribute name](#). The value is implicitly the empty string.

EXAMPLE 635

In the following example, the disabled attribute is given with the empty attribute syntax:

```
<input disabled>
```

If an attribute using the empty attribute syntax is to be followed by another attribute, then there must be a [space character](#) separating the two.

Unquoted attribute value syntax

The [attribute name](#), followed by zero or more [space characters](#), followed by a single U+003D EQUALS SIGN character, followed by zero or more [space characters](#), followed by the [attribute value](#), which, in addition to the requirements given above for attribute values, must not contain any literal [space characters](#), any U+0022 QUOTATION MARK characters ("), U+0027 APOSTROPHE characters ('), U+003D EQUALS SIGN characters (=), U+003C LESS-THAN SIGN characters (<), U+003E GREATER-THAN SIGN characters (>), or U+0060 GRAVE ACCENT characters (`), and must not be the empty string.

EXAMPLE 636

In the following example, the value attribute is given with the unquoted attribute value syntax:

```
<input value=yes>
```

If an attribute using the unquoted attribute syntax is to be followed by another attribute or by the optional U+002F SOLIDUS character (/) allowed in step 6 of the [start tag](#) syntax above, then there must be a [space character](#) separating the two.

Single-quoted attribute value syntax

The [attribute name](#), followed by zero or more [space characters](#), followed by a single U+003D EQUALS SIGN character, followed by zero or more [space characters](#), followed by a single U+0027 APOSTROPHE character ('), followed by the [attribute value](#), which, in addition to the requirements given above for attribute values, must not contain any literal U+0027 APOSTROPHE characters ('), and finally followed by a second single U+0027 APOSTROPHE character (').

EXAMPLE 637

In the following example, the type attribute is given with the single-quoted attribute value syntax:

```
<input type='checkbox'>
```


If an attribute using the single-quoted attribute syntax is to be followed by another attribute, then there must be a [space character](#) separating the two.

Double-quoted attribute value syntax

The [attribute name](#), followed by zero or more [space characters](#), followed by a single U+003D EQUALS SIGN character, followed by zero or more [space characters](#), followed by a single U+0022 QUOTATION MARK character ("), followed by the [attribute value](#), which, in addition to the requirements given above for attribute values, must not contain any literal U+0022 QUOTATION MARK characters ("), and finally followed by a second single U+0022 QUOTATION MARK character (").

EXAMPLE 638

In the following example, the name attribute is given with the double-quoted attribute value syntax:

```
<input name="be evil">
```

If an attribute using the double-quoted attribute syntax is to be followed by another attribute, then there must be a [space character](#) separating the two.

There must never be two or more attributes on the same start tag whose names are an [ASCII case-insensitive](#) match for each other.



When a [foreign element](#) has one of the namespaced attributes given by the local name and namespace of the first and second cells of a row from the following table, it must be written using the name given by the third cell from the same row.

Local name	Namespace	Attribute name
actuate	XLink namespace	xlink:actuate
arcrole	XLink namespace	xlink:arcrole
href	XLink namespace	xlink:href
role	XLink namespace	xlink:role
show	XLink namespace	xlink:show
title	XLink namespace	xlink:title
type	XLink namespace	xlink:type
base	XML namespace	xml:base
lang	XML namespace	xml:lang
space	XML namespace	xml:space
xmlns	XMLNS namespace	xmlns
xlink	XMLNS namespace	xmlns:xlink

No other namespaced attribute can be expressed in [the HTML syntax](#).

NOTE:

Whether the attributes in the table above are conforming or not is defined by other specifications (e.g., the SVG and MathML specifications); this section only describes the syntax rules if the attributes are serialized using the HTML syntax.

8.1.2.4. Optional tags

Certain tags can be **omitted**.

NOTE:

Omitting an element's [start tag](#) in the situations described below does not mean the element is not present; it is implied, but it is still there. For example, an HTML document always has a root [<html>](#) element, even if the string `<html>` doesn't appear anywhere in the markup.

An [<html>](#) element's [start tag](#) may be omitted if the first thing inside the [<html>](#) element is not a [comment](#).

EXAMPLE 639

For example, in the following case it's ok to remove the "<html>" tag:

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Hello</title>
</head>
<body>
  <p>Welcome to this example.</p>
</body>
</html>
```

Doing so would make the document look like this:

```
<!DOCTYPE HTML>

<head>
  <title>Hello</title>
</head>
<body>
  <p>Welcome to this example.</p>
</body>
</html>
```

This has the exact same DOM. In particular, note that white space around the root element is ignored by the parser. The following example would also have the exact same DOM:

```
<!DOCTYPE HTML><head>
  <title>Hello</title>
</head>
<body>
  <p>Welcome to this example.</p>
</body>
</html>
```

However, in the following example, removing the start tag moves the comment to before the [<html>](#) element:

```
<!DOCTYPE HTML>
<html>
<!-- where is this comment in the DOM? -->
<head>
  <title>Hello</title>
</head>
<body>
  <p>Welcome to this example.</p>
</body>
</html>
```

With the tag removed, the document actually turns into the same as this:

```
<!DOCTYPE HTML>
<!-- where is this comment in the DOM? -->
<html>
<head>
  <title>Hello</title>
</head>
<body>
  <p>Welcome to this example.</p>
</body>
</html>
```

This is why the tag can only be removed if it is not followed by a comment: removing the tag when there is a comment there changes the document's resulting parse tree. Of course, if the position of the comment does not matter, then the tag can be omitted, as if the comment had been moved to before the start tag in the first place.

An [<html>](#) element's [end tag](#) may be omitted if the [<html>](#) element is not immediately followed by a [comment](#).

A [<head>](#) element's [start tag](#) may be omitted if the element is empty, or if the first thing inside the [<head>](#) element is an element.

A [<head>](#) element's [end tag](#) may be omitted if the [<head>](#) element is not immediately followed by a [space character](#) or a [comment](#).

A [<body>](#) element's [start tag](#) may be omitted if the element is empty, or if the first thing inside the [<body>](#) element is not a [space character](#) or a [comment](#), except if the first thing inside the [<body>](#) element is a [<meta>](#), [<link>](#), [<script>](#), [<style>](#), or [<template>](#) element.

A [<body>](#) element's [end tag](#) may be omitted if the [<body>](#) element is not immediately followed by a [comment](#).

EXAMPLE 640

Note that in the example above, the `<head>` element start and end tags, and the `<body>` element start tag, can't be omitted, because they are surrounded by white space:

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Hello</title>
</head>
<body>
  <p>Welcome to this example.</p>
</body>
</html>
```

(The body and `<html>` element end tags could be omitted without trouble; any spaces after those get parsed into the `<body>` element anyway.)

Usually, however, white space isn't an issue. If we first remove the white space we don't care about:

```
<!DOCTYPE HTML><html><head><title>Hello</title></head><body><p>Welcome to this example.
</p></body></html>
```

Then we can omit a number of tags without affecting the DOM:

```
<!DOCTYPE HTML><title>Hello</title><p>Welcome to this example.</p>
```

At that point, we can also add some white space back:

```
<!DOCTYPE HTML>
<title>Hello</title>
<p>Welcome to this example.</p>
```

This would be equivalent to this document, with the omitted tags shown in their parser-implied positions; the only white space text node that results from this is the newline at the end of the `<head>` element:

```
<!DOCTYPE HTML>
<html><head><title>Hello</title>
</head><body><p>Welcome to this example.</p></body></html>
```

An `<li>` element's **end tag** may be omitted if the `<li>` element is immediately followed by another `<li>` element or if there is no more content in the parent element.

A `<dt>` element's **end tag** may be omitted if the `<dt>` element is immediately followed by another `<dt>` element or a `<dd>` element.

A `<dd>` element's **end tag** may be omitted if the `<dd>` element is immediately followed by another `<dd>` element or a `<dt>` element, or if there is no more content in the parent element.

A `<p>` element's **end tag** may be omitted if the `<p>` element is immediately followed by an `<address>`, `<article>`, `<aside>`, `<blockquote>`, `<details>`, `<div>`, `<dl>`, `<fieldset>`, `<figcaption>`, `<figure>`, `<footer>`, `<form>`, `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`, `<header>`, `<hr>`, `<main>`, `<menu>`, `<nav>`, `<ol>`, `<p>`, `<pre>`, `<section>`, `<table>`, or `<ul>` element, or if there is no more content in the parent element and the parent element is an **HTML element** that is not an `<a>`, `<audio>`, `<del>`, `<ins>`, `<map>`, `<noscript>`, or `<video>` element.

EXAMPLE 641

We can thus simplify the earlier example further:

```
<!DOCTYPE HTML><title>Hello</title><p>Welcome to this example.</p>
```

An [<rt>](#) element's [end tag](#) may be omitted if the [<rt>](#) element is immediately followed by an [rt](#) or [<rp>](#) element, or if there is no more content in the parent element.

An [<rp>](#) element's [end tag](#) may be omitted if the [<rp>](#) element is immediately followed by an [rt](#) or [<rp>](#) element, or if there is no more content in the parent element.

An [<optgroup>](#) element's [end tag](#) may be omitted if the [<optgroup>](#) element is immediately followed by another [<optgroup>](#) element, or if there is no more content in the parent element.

An [<option>](#) element's [end tag](#) may be omitted if the [<option>](#) element is immediately followed by another [<option>](#) element, or if it is immediately followed by an [<optgroup>](#) element, or if there is no more content in the parent element.

A [<colgroup>](#) element's [start tag](#) may be omitted if the first thing inside the [<colgroup>](#) element is a [<col>](#) element, and if the element is not immediately preceded by another [<colgroup>](#) element whose [end tag](#) has been omitted. (It can't be omitted if the element is empty.)

A [<colgroup>](#) element's [end tag](#) may be omitted if the [<colgroup>](#) element is not immediately followed by a [space character](#) or a [comment](#).

A [<caption>](#) element's [end tag](#) may be omitted if the [<caption>](#) element is not immediately followed by a [space character](#) or a [comment](#).

A [<thead>](#) element's [end tag](#) may be omitted if the [<thead>](#) element is immediately followed by a [tbody](#) or [<tfoot>](#) element.

A [<tbody>](#) element's [start tag](#) may be omitted if the first thing inside the [<tbody>](#) element is a [<tr>](#) element, and if the element is not immediately preceded by a [<tbody>](#), [<thead>](#), or [<tfoot>](#) element whose [end tag](#) has been omitted. (It can't be omitted if the element is empty.)

A [<tbody>](#) element's [end tag](#) may be omitted if the [<tbody>](#) element is immediately followed by a [tbody](#) or [<tfoot>](#) element, or if there is no more content in the parent element.

A [<tfoot>](#) element's [end tag](#) may be omitted if the [<tfoot>](#) element is immediately followed by a [<tbody>](#) element, or if there is no more content in the parent element.

A [<tr>](#) element's [end tag](#) may be omitted if the [<tr>](#) element is immediately followed by another [<tr>](#) element, or if there is no more content in the parent element.

A [<td>](#) element's [end tag](#) may be omitted if the [<td>](#) element is immediately followed by a [td](#) or [<th>](#) element, or if there is no more content in the parent element.

A [<th>](#) element's [end tag](#) may be omitted if the [<th>](#) element is immediately followed by a [td](#) or [<th>](#) element, or if there is no more content in the parent element.

EXAMPLE 642

The ability to omit all these table-related tags makes table markup much terser.

Take this example:

```
<table>
<caption>37547 TEE Electric Powered Rail Car Train Functions (Abbreviated)</caption>
<colgroup><col><col><col></colgroup>
<thead>
<tr>
  <th>Function</th>
  <th>Control Unit</th>
  <th>Central Station</th>
</tr>
</thead>
<tbody>
<tr>
  <td>Headlights</td>
  <td>✓</td>
  <td>✓</td>
</tr>
<tr>
  <td>Interior Lights</td>
  <td>✓</td>
  <td>✓</td>
</tr>
<tr>
  <td>Electric locomotive operating sounds</td>
  <td>✓</td>
  <td>✓</td>
</tr>
<tr>
  <td>Engineer's cab lighting</td>
  <td></td>
  <td>✓</td>
</tr>
<tr>
  <td>Station Announcements - Swiss</td>
  <td></td>
  <td>✓</td>
</tr>
</tbody>
</table>
```

The exact same table, modulo some white space differences, could be marked up as follows:

```

<table>
<caption>37547 TEE Electric Powered Rail Car Train Functions (Abbreviated)
<colgroup><col><col><col>
<thead>
<tr>
  <th>Function
  <th>Control Unit
  <th>Central Station
</thead>
<tbody>
<tr>
  <td>Headlights
  <td>✓
  <td>✓
</tr>
<tr>
  <td>Interior Lights
  <td>✓
  <td>✓
</tr>
<tr>
  <td>Electric locomotive operating sounds
  <td>✓
  <td>✓
</tr>
<tr>
  <td>Engineer's cab lighting
  <td>
  <td>✓
</tr>
<tr>
  <td>Station Announcements - Swiss
  <td>
  <td>✓
</tr>
</table>

```

Since the cells take up much less room this way, this can be made even terser by having each row on one line:

```

<table>
<caption>37547 TEE Electric Powered Rail Car Train Functions (Abbreviated)
<colgroup><col><col><col>
<thead>
<tr> <th>Function                <th>Control Unit    <th>Central Station
</thead>
<tbody>
<tr> <td>Headlights              <td>✓              <td>✓
<tr> <td>Interior Lights         <td>✓              <td>✓
<tr> <td>Electric locomotive operating sounds <td>✓              <td>✓
<tr> <td>Engineer's cab lighting <td>                <td>✓
<tr> <td>Station Announcements - Swiss <td>                <td>✓
</tbody>
</table>

```

The only differences between these tables, at the DOM level, is with the precise position of the (in any case semantically-neutral) white space.

However, a [start tag](#) must never be omitted if it has any attributes.

EXAMPLE 643

Returning to the earlier example with all the white space removed and then all the optional tags removed:

```
<!DOCTYPE HTML><title>Hello</title><p>Welcome to this example.
```

If the `<body>` element in this example had to have a `class` attribute and the `<html>` element had to have a `lang` attribute, the markup would have to become:

```
<!DOCTYPE HTML><html lang="en"><title>Hello</title><body class="demo"><p>Welcome to this example.
```

NOTE:

This section assumes that the document is conforming, in particular, that there are no [content model](#) violations. Omitting tags in the fashion described in this section in a document that does not conform to the [content models](#) described in this specification is likely to result in unexpected DOM differences (this is, in part, what the content models are designed to avoid).

§ 8.1.2.5. *Restrictions on content models*

For historical reasons, certain elements have extra restrictions beyond even the restrictions given by their content model.

A `<table>` element must not contain `<tr>` elements, even though these elements are technically allowed inside `<table>` elements according to the content models described in this specification. (If a `<tr>` element is put inside a `table` in the markup, it will in fact imply a `tbody` start tag before it.)

A single [newline](#) may be placed immediately after the [start tag](#) of `pre` and `<textarea>` elements. This does not affect the processing of the element. The otherwise optional [newline](#) *must* be included if the element's contents themselves start with a [newline](#) (because otherwise the leading newline in the contents would be treated like the optional newline, and ignored).

EXAMPLE 644

The following two `pre` blocks are equivalent:

```
<pre>Hello</pre>
```

```
<pre>
Hello</pre>
```

§ 8.1.2.6. *Restrictions on the contents of raw text and escapable raw text elements*

The text in [raw text](#) and [escapable raw text elements](#) must not contain any occurrences of the string `"</"` (U+003C LESS-THAN SIGN, U+002F SOLIDUS) followed by characters that case-insensitively match the tag name of the element followed by one of U+0009 CHARACTER TABULATION (tab), U+000A LINE FEED (LF), U+000C FORM FEED (FF), U+000D CARRIAGE RETURN (CR), U+0020 SPACE, U+003E GREATER-THAN SIGN (`>`), or U+002F SOLIDUS (`/`).

§ 8.1.3. **Text**

Text is allowed inside elements, attribute values, and comments. Extra constraints are placed on what is and what is not allowed in text based on where the text is to be put, as described in the other sections.

§ 8.1.3.1. *Newlines*

Newlines in HTML may be represented either as U+000D CARRIAGE RETURN (CR) characters, U+000A LINE FEED (LF) characters, or pairs of U+000D CARRIAGE RETURN (CR), U+000A LINE FEED (LF) characters in that order.

Where [character references](#) are allowed, a character reference of a U+000A LINE FEED (LF) character (but not a U+000D CARRIAGE RETURN (CR) character) also represents a [newline](#).

§ 8.1.4. Character references

In certain cases described in other sections, [text](#) may be mixed with **character references**. These can be used to escape characters that couldn't otherwise legally be included in [text](#).

Character references must start with a U+0026 AMPERSAND character (&). Following this, there are three possible kinds of character references:

Named character references

The ampersand must be followed by one of the names given in [§8.5 Named character references](#) section, using the same case. The name must be one that is terminated by a U+003B SEMICOLON character (;).

Decimal numeric character reference

The ampersand must be followed by a U+0023 NUMBER SIGN character (#), followed by one or more [ASCII digits](#), representing a base-ten integer that corresponds to a Unicode code point that is allowed according to the definition below. The digits must then be followed by a U+003B SEMICOLON character (;).

Hexadecimal numeric character reference

The ampersand must be followed by a U+0023 NUMBER SIGN character (#), which must be followed by either a U+0078 LATIN SMALL LETTER X character (x) or a U+0058 LATIN CAPITAL LETTER X character (X), which must then be followed by one or more [ASCII hex digits](#), representing a hexadecimal integer that corresponds to a Unicode code point that is allowed according to the definition below. The digits must then be followed by a U+003B SEMICOLON character (;).

The numeric character reference forms described above are allowed to reference any Unicode code point other than U+0000, U+000D, permanently undefined Unicode characters (noncharacters), surrogates (U+D800–U+DFFF), and [control characters](#) other than [space characters](#).

An **ambiguous ampersand** is a U+0026 AMPERSAND character (&) that is followed by one or more [alphanumeric ASCII characters](#), followed by a U+003B SEMICOLON character (;), where these characters do not match any of the names given in the [§8.5 Named character references](#) section.

§ 8.1.5. CDATA sections

CDATA sections must consist of the following components, in this order:

1. The string "<![CDATA[".
2. Optionally, [text](#), with the additional restriction that the text must not contain the string "]]>".
3. The string "]]>".

EXAMPLE 645

CDATA sections can only be used in foreign content (MathML or SVG). In this example, a CDATA section is used to escape the contents of an `<ms>` element:

```
<p>You can add a string to a number, but this stringifies the number:</p>
<math>
<ms><![CDATA[x<y]]></ms>
<mo>+</mo>
<mn>3</mn>
<mo>=</mo>
<ms><![CDATA[x<y3]]></ms>
</math>
```

§ 8.1.6. Comments

Comments must start with the four character sequence U+003C LESS-THAN SIGN, U+0021 EXCLAMATION MARK, U+002D HYPHEN-MINUS, U+002D HYPHEN-MINUS (<!--). Following this sequence, the comment may have [text](#), with the additional restriction that the text must not start with a single U+003E GREATER-THAN SIGN character (>), nor start with a U+002D HYPHEN-MINUS character (-) followed by a U+003E GREATER-THAN SIGN (>) character, nor contain two consecutive U+002D HYPHEN-MINUS characters (--), nor end with a U+002D HYPHEN-MINUS character (-). Finally, the comment must be ended by the three character sequence U+002D HYPHEN-MINUS, U+002D HYPHEN-MINUS, U+003E GREATER-THAN SIGN (-->).

§ 8.2. Parsing HTML documents

This section only applies to user agents, data mining tools, and conformance checkers.

NOTE:

The rules for parsing XML documents into DOM trees are covered by the next section, entitled "[the XHTML syntax](#)".

User agents must use the parsing rules described in this section to generate the DOM trees from [text/html](#) resources. Together, these rules define what is referred to as the **HTML parser**.

While the HTML syntax described in this specification bears a close resemblance to SGML and XML, it is a separate language with its own parsing rules.

Some earlier versions of HTML (in particular from HTML 2.0 to HTML 4.01) were based on SGML and used SGML parsing rules. However, few (if any) web browsers ever implemented true SGML parsing for HTML documents; the only user agents to strictly handle HTML as an SGML application have historically been validators. The resulting confusion — with validators claiming documents to have one representation while widely deployed Web browsers interoperably implemented a different representation — has wasted decades of productivity. This version of HTML thus returns to a non-SGML basis.

Authors interested in using SGML tools in their authoring pipeline are encouraged to use XML tools and the XML serialization of HTML.

This specification defines the parsing rules for HTML documents, whether they are syntactically correct or not. Certain points in the parsing algorithm are said to be **parse errors**. The error handling for parse errors is well-defined (that's the processing rules described throughout this specification), but user agents, while parsing an HTML document, may [abort the parser](#) at the first [parse error](#) that they encounter for which they do not wish to apply the rules described in this specification.

Conformance checkers must report at least one [parse error](#) condition to the user if one or more [parse error](#) conditions exist in the document and must not report [parse error](#) conditions if none exist in the document. Conformance checkers may report more than one [parse error](#) condition if more than one [parse error](#) condition exists in the document.

NOTE:

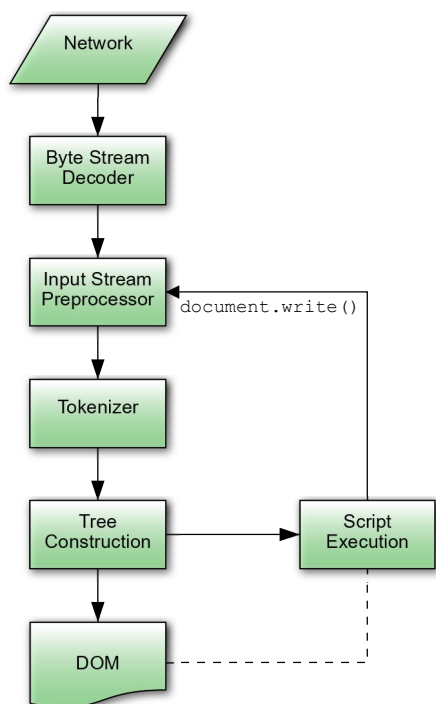
[Parse errors](#) are only errors with the syntax of HTML. In addition to checking for [parse errors](#), conformance checkers will also verify that the document obeys all the other conformance requirements described in this specification.

For the purposes of conformance checkers, if a resource is determined to be in [the HTML syntax](#), then it is an [HTML document](#).

NOTE:

As stated in the terminology section, references to [element types](#) that do not explicitly specify a namespace always refer to elements in the [HTML namespace](#). For example, if the spec talks about "a [<menuitem>](#) element", then that is an element with the local name "[<menuitem>](#)", the namespace "[http://www.w3.org/1999/xhtml](#)", and the interface [HTMLMenuItemElement](#). Where possible, references to such elements are hyperlinked to their definition.

§ 8.2.1. Overview of the parsing model



The input to the HTML parsing process consists of a stream of [Unicode code points](#), which is passed through a [tokenization](#) stage followed by a tree construction stage. The output is a [Document](#) object.

NOTE:

Implementations that [do not support scripting](#) do not have to actually create a DOM Document object, but the DOM tree in such cases is still used as the model for the rest of the specification.

In the common case, the data handled by the tokenization stage comes from the network, but [it can also come from script](#) running in the user agent, e.g., using the [document.write\(\)](#) API.

There is only one set of states for the tokenizer stage and the tree construction stage, but the tree construction stage is **reentrant**, meaning that while the tree construction stage is handling one token, the tokenizer might be resumed, causing further tokens to be emitted and processed before the first token's processing is complete.

EXAMPLE 646

In the following example, the tree construction stage will be called upon to handle a "p" start tag token while handling the "script" end tag token:

```

...
<script>
document.write('<p>');
</script>
...

```

To handle these cases, parsers have a **script nesting level**, which must be initially set to zero, and a **parser pause flag**, which must be initially set to false.

§ 8.2.2. The input byte stream

The stream of Unicode code points that comprises the input to the tokenization stage will be initially seen by the user agent as a stream of bytes (typically coming over the network or from the local file system). The bytes encode the actual characters according to a particular *character encoding*, which the user agent uses to decode the bytes into characters.

NOTE:

For XML documents, the algorithm user agents are required to use to determine the character encoding is given by the XML specification. This section does not apply to XML documents. [\[XML\]](#)

Usually, the [encoding sniffing algorithm](#) defined below is used to determine the character encoding.

Given a character encoding, the bytes in the [input byte stream](#) must be converted to characters for the tokenizer's [input stream](#), by passing the [input byte stream](#) and character encoding to [decode](#).

NOTE:

A leading Byte Order Mark (BOM) causes the character encoding argument to be ignored and will itself be skipped.

NOTE:

Bytes or sequences of bytes in the original byte stream that did not conform to the Encoding standard (e.g., invalid UTF-8 byte sequences in a UTF-8 input byte stream) are errors that conformance checkers are expected to report.

[\[ENCODING\]](#)

⚠Warning! The decoder algorithms describe how to handle invalid input; for security reasons, it is imperative that those rules be followed precisely. Differences in how invalid byte sequences are handled can result in, amongst other problems, script injection vulnerabilities ("XSS").

When the HTML parser is decoding an input byte stream, it uses a character encoding and a **confidence**. The confidence is either *tentative*, *certain*, or *irrelevant*. The encoding used, and whether the confidence in that encoding is *tentative* or *certain*, is [used during the parsing](#) to determine whether to [change the encoding](#). If no encoding is necessary, e.g., because the parser is operating on a Unicode stream and doesn't have to use a character encoding at all, then the [confidence](#) is *irrelevant*.

NOTE:

Some algorithms feed the parser by directly adding characters to the [input stream](#) rather than adding bytes to the [input byte stream](#).

§ 8.2.2.1. Parsing with a known character encoding

When the HTML parser is to operate on an input byte stream that has a **known definite encoding**, then the character encoding is that encoding and the [confidence](#) is *certain*.

§ 8.2.2.2. Determining the character encoding

In some cases, it might be impractical to unambiguously determine the encoding before parsing the document. Because of this, this specification provides for a two-pass mechanism with an optional pre-scan. Implementations are allowed, as described below, to apply a simplified parsing algorithm to whatever bytes they have available before beginning to parse the document. Then, the real parser is started, using a tentative encoding derived from this pre-parse and other out-of-band metadata. If, while the document is being loaded, the user agent discovers a character encoding declaration that conflicts with this information, then the parser can get reinvoked to perform a parse of the document with the real encoding.

User agents must use the following algorithm, called the **encoding sniffing algorithm**, to determine the character encoding to use when decoding a document in the first pass. This algorithm takes as input any out-of-band metadata available to the user agent (e.g., the [Content-Type metadata](#) of the document) and all the bytes available so far, and returns a character encoding and a [confidence](#) that is either *tentative* or *certain*.

1. If the user has explicitly instructed the user agent to override the document's character encoding with a specific encoding, optionally return that encoding with the [confidence](#) *certain* and abort these steps.

NOTE:

Typically, user agents remember such user requests across sessions, and in some cases apply them to documents in iframes as well.

2. The user agent may wait for more bytes of the resource to be available, either in this step or at any later step in this algorithm. For instance, a user agent might wait 500ms or 1024 bytes, whichever came first. In general preparsing the source to find the encoding improves performance, as it reduces the need to throw away the data structures used when parsing upon finding the encoding information. However, if the user agent delays too long to obtain data to determine the encoding, then the cost of the delay could outweigh any performance improvements from the preparsing.

NOTE:

The authoring conformance requirements for character encoding declarations limit them to only appearing [in the first 1024 bytes](#). User agents are therefore encouraged to use the prescan algorithm below (as invoked by these steps) on the first 1024 bytes, but not to stall beyond that.

3. If the transport layer specifies a character encoding, and it is supported, return that encoding with the [confidence](#) *certain*, and abort these steps.
4. Optionally [prescan the byte stream to determine its encoding](#). The *end condition* is that the user agent decides that scanning further bytes would not be efficient. User agents are encouraged to only prescan the first 1024 bytes. User agents may decide that scanning *any* bytes is not efficient, in which case these substeps are entirely skipped.

The aforementioned algorithm either aborts unsuccessfully or returns a character encoding. If it returns a character encoding, then this algorithm must be aborted, returning the same encoding, with [confidence](#) *tentative*.

5. If the [HTML parser](#) for which this algorithm is being run is associated with a [Document](#) that is itself in a [nested browsing context](#), run these substeps:
 1. Let *new document* be the Document with which the [HTML parser](#) is associated.
 2. Let *parent document* be the Document [through which](#) *new document* is nested (the [active document](#) of the [parent browsing context](#) of *new document*).
 3. If *parent document*'s [origin](#) is not the [same origin](#) as *new document*'s [origin](#), then abort these substeps.
 4. If *parent document*'s [character encoding](#) is not an [ASCII-compatible encoding](#), then abort these substeps.
 5. Return *parent document*'s [character encoding](#), with the [confidence](#) *tentative*, and abort the [encoding sniffing algorithm](#)'s steps.
6. Otherwise, if the user agent has information on the likely encoding for this page, e.g., based on the encoding of the page when it was last visited, then return that encoding, with the [confidence](#) *tentative*, and abort these steps.
7. The user agent may attempt to autodetect the character encoding from applying frequency analysis or other algorithms to the data stream. Such algorithms may use information about the resource other than the resource's contents, including the address of the resource. If autodetection succeeds in determining a character encoding, and that encoding is a supported encoding, then return that encoding, with the [confidence](#) *tentative*, and abort these steps. [\[UNIVCHARDET\]](#)

NOTE:

User agents are generally discouraged from attempting to autodetect encodings for resources obtained over the network, since doing so involves inherently non-interoperable heuristics. Attempting to detect encodings based on an HTML document's preamble is especially tricky since HTML markup typically uses only ASCII characters, and HTML documents tend to begin with a lot of markup rather than with text content.

NOTE:

The UTF-8 encoding has a highly detectable bit pattern. Files from the local file system that contain bytes with values greater than 0x7F which match the UTF-8 pattern are very likely to be UTF-8, while documents with byte sequences that do not match it are very likely not. When a user agent can examine the whole file, rather than just the preamble, detecting for UTF-8 specifically can be especially effective. [\[PPUTF8\]](#) [\[UTF8DET\]](#)

8. Otherwise, return an implementation-defined or user-specified default character encoding, with the [confidence](#) *tentative*.

In controlled environments or in environments where the encoding of documents can be prescribed (for example, for user agents intended for dedicated use in new networks), the comprehensive UTF-8 encoding is suggested.

In other environments, the default encoding is typically dependent on the user’s locale (an approximation of the languages, and thus often encodings, of the pages that the user is likely to frequent). The following table gives suggested defaults based on the user’s locale, for compatibility with legacy content. Locales are identified by BCP 47 language tags. [\[BCP47\]](#) [\[ENCODING\]](#)

Locale language		Suggested default encoding
ar	Arabic	windows-1256
ba	Bashkir	windows-1251
be	Belarusian	windows-1251
bg	Bulgarian	windows-1251
cs	Czech	windows-1250
el	Greek	ISO-8859-7
et	Estonian	windows-1257
fa	Persian	windows-1256
he	Hebrew	windows-1255
hr	Croatian	windows-1250
hu	Hungarian	ISO-8859-2
ja	Japanese	Shift_JIS
kk	Kazakh	windows-1251
ko	Korean	euc-kr
ku	Kurdish	windows-1254
ky	Kyrgyz	windows-1251
lt	Lithuanian	windows-1257
lv	Latvian	windows-1257
mk	Macedonian	windows-1251
pl	Polish	ISO-8859-2
ru	Russian	windows-1251
sah	Yakut	windows-1251
sk	Slovak	windows-1250
sl	Slovenian	ISO-8859-2
sr	Serbian	windows-1251
tg	Tajik	windows-1251
th	Thai	windows-874
tr	Turkish	windows-1254
tt	Tatar	windows-1251
uk	Ukrainian	windows-1251
vi	Vietnamese	windows-1258
zh-CN	Chinese (People’s Republic of China)	GB18030
zh-TW	Chinese (Taiwan)	Big5
All other locales		windows-1252

The contents of this table are derived from the intersection of Windows, Chrome, and Firefox defaults.

The [document’s character encoding](#) must immediately be set to the value returned from this algorithm, at the same time as the user agent uses the returned value to select the decoder to use for the input byte stream.



When an algorithm requires a user agent to **prescan a byte stream to determine its encoding**, given some defined *end condition*, then it must run the following steps. These steps either abort unsuccessfully or return a character encoding. If at any point during these steps (including during instances of the [get an attribute](#) algorithm invoked by this one) the user agent either runs out of bytes (meaning the *position* pointer created in the first step below goes beyond the end of the byte stream obtained so far) or reaches its *end condition*, then abort the [prescan a byte stream to determine its encoding](#) algorithm unsuccessfully.

1. Let *position* be a pointer to a byte in the input byte stream, initially pointing at the first byte.

2. *Loop*: If *position* points to:

↪ **A sequence of bytes starting with: 0x3C 0x21 0x2D 0x2D (ASCII '<!--')**

Advance the *position* pointer so that it points at the first 0x3E byte which is preceded by two 0x2D bytes (i.e., at the end of an ASCII '->' sequence) and comes after the 0x3C byte that was found. (The two 0x2D bytes can be the same as those in the '<!--' sequence.)

↪ **A sequence of bytes starting with: 0x3C, 0x4D or 0x6D, 0x45 or 0x65, 0x54 or 0x74, 0x41 or 0x61, and one of 0x09, 0x0A, 0x0C, 0x0D, 0x20, 0x2F (case-insensitive ASCII '<meta' followed by a space or slash)**

1. Advance the *position* pointer so that it points at the next 0x09, 0x0A, 0x0C, 0x0D, 0x20, or 0x2F byte (the one in sequence of characters matched above).
2. Let *attribute list* be an empty list of strings.
3. Let *got pragma* be false.
4. Let *need pragma* be null.
5. Let *charset* be the null value (which, for the purposes of this algorithm, is distinct from an unrecognized encoding or the empty string).
6. *Attributes*: [Get an attribute](#) and its value. If no attribute was computed, then jump to the *processing* step below.
7. If the attribute's name is already in *attribute list*, then return to the step labeled *attributes*.
8. Add the attribute's name to *attribute list*.
9. Run the appropriate step from the following list, if one applies:

↪ **If the attribute's name is "http-equiv"**

If the attribute's value is "content-type", then set *got pragma* to true.

↪ **If the attribute's name is "content"**

Apply the [algorithm for extracting a character encoding from a meta element](#), giving the attribute's value as the string to parse. If a character encoding is returned, and if *charset* is still set to null, let *charset* be the encoding returned, and set *need pragma* to true.

↪ **If the attribute's name is "charset"**

Let *charset* be the result of [getting an encoding](#) from the attribute's value, and set *need pragma* to false.

10. Return to the step labeled *attributes*.
11. *Processing*: If *need pragma* is null, then jump to the step below labeled *next byte*.
12. If *need pragma* is true but *got pragma* is false, then jump to the step below labeled *next byte*.
13. If *charset* is failure, then jump to the step below labeled *next byte*.
14. If *charset* is a [UTF-16 encoding](#), then set *charset* to [UTF-8](#).
15. If *charset* is [x-user-defined](#), then set *charset* to [windows-1252](#).
16. Abort the [prescan a byte stream to determine its encoding](#) algorithm, returning the encoding given by *charset*.

↪ **A sequence of bytes starting with a 0x3C byte (ASCII <), optionally a 0x2F byte (ASCII /), and finally a byte in the range 0x41-0x5A or 0x61-0x7A (an ASCII letter)**

1. Advance the *position* pointer so that it points at the next 0x09 (ASCII TAB), 0x0A (ASCII LF), 0x0C (ASCII FF), 0x0D (ASCII CR), 0x20 (ASCII space), or 0x3E (ASCII >) byte.
2. Repeatedly [get an attribute](#) until no further attributes can be found, then jump to the step below labeled *next byte*.

↪ **A sequence of bytes starting with: 0x3C 0x21 (ASCII '<')**

↪ **A sequence of bytes starting with: 0x3C 0x2F (ASCII '</')**

↪ **A sequence of bytes starting with: 0x3C 0x3F (ASCII '<?')**

Advance the *position* pointer so that it points at the first 0x3E byte (ASCII >) that comes after the 0x3C byte that was found.

↪ **Any other byte**

Do nothing with that byte.

3. *Next byte*: Move *position* so it points at the next byte in the input byte stream, and return to the step above labeled *loop*.

When the [prescan a byte stream to determine its encoding](#) algorithm says to **get an attribute**, it means doing this:

1. If the byte at *position* is one of 0x09 (ASCII TAB), 0x0A (ASCII LF), 0x0C (ASCII FF), 0x0D (ASCII CR), 0x20 (ASCII space), or 0x2F (ASCII /) then advance *position* to the next byte and redo this step.
2. If the byte at *position* is 0x3E (ASCII >), then abort the [get an attribute](#) algorithm. There isn't one.
3. Otherwise, the byte at *position* is the start of the attribute name. Let *attribute name* and *attribute value* be the empty string.
4. Process the byte at *position* as follows:

↪ **If it is 0x3D (ASCII =), and the *attribute name* is longer than the empty string**

Advance *position* to the next byte and jump to the step below labeled *value*.

↪ **If it is 0x09 (ASCII TAB), 0x0A (ASCII LF), 0x0C (ASCII FF), 0x0D (ASCII CR), or 0x20 (ASCII space)**

Jump to the step below labeled *spaces*.

↪ **If it is 0x2F (ASCII /) or 0x3E (ASCII >)**

Abort the [get an attribute](#) algorithm. The attribute's name is the value of *attribute name*, its value is the empty string.

↪ **If it is in the range 0x41 (ASCII A) to 0x5A (ASCII Z)**

Append the Unicode character with code point $b+0x20$ to *attribute name* (where b is the value of the byte at *position*). (This converts the input to lowercase.)

↪ **Anything else**

Append the Unicode character with the same code point as the value of the byte at *position* to *attribute name*. (It doesn't actually matter how bytes outside the ASCII range are handled here, since only ASCII characters can contribute to the detection of a character encoding.)

5. Advance *position* to the next byte and return to the previous step.
6. *Spaces*: If the byte at *position* is one of 0x09 (ASCII TAB), 0x0A (ASCII LF), 0x0C (ASCII FF), 0x0D (ASCII CR), or 0x20 (ASCII space) then advance *position* to the next byte, then, repeat this step.
7. If the byte at *position* is not 0x3D (ASCII =), abort the [get an attribute](#) algorithm. The attribute's name is the value of *attribute name*, its value is the empty string.
8. Advance *position* past the 0x3D (ASCII =) byte.
9. *Value*: If the byte at *position* is one of 0x09 (ASCII TAB), 0x0A (ASCII LF), 0x0C (ASCII FF), 0x0D (ASCII CR), or 0x20 (ASCII space) then advance *position* to the next byte, then, repeat this step.

10. Process the byte at *position* as follows:

↪ **If it is 0x22 (ASCII ") or 0x27 (ASCII ')**

1. Let *b* be the value of the byte at *position*.
2. *Quote loop*: Advance *position* to the next byte.
3. If the value of the byte at *position* is the value of *b*, then advance *position* to the next byte and abort the "get an attribute" algorithm. The attribute's name is the value of *attribute name*, and its value is the value of *attribute value*.
4. Otherwise, if the value of the byte at *position* is in the range 0x41 (ASCII A) to 0x5A (ASCII Z), then append a Unicode character to *attribute value* whose code point is 0x20 more than the value of the byte at *position*.
5. Otherwise, append a Unicode character to *attribute value* whose code point is the same as the value of the byte at *position*.
6. Return to the step above labeled *quote loop*.

↪ **If it is 0x3E (ASCII >)**

Abort the [get an attribute](#) algorithm. The attribute's name is the value of *attribute name*, its value is the empty string.

↪ **If it is in the range 0x41 (ASCII A) to 0x5A (ASCII Z)**

Append the Unicode character with code point *b*+0x20 to *attribute value* (where *b* is the value of the byte at *position*). Advance *position* to the next byte.

↪ **Anything else**

Append the Unicode character with the same code point as the value of the byte at *position* to *attribute value*. Advance *position* to the next byte.

11. Process the byte at *position* as follows:

↪ **If it is 0x09 (ASCII TAB), 0x0A (ASCII LF), 0x0C (ASCII FF), 0x0D (ASCII CR), 0x20 (ASCII space), or 0x3E (ASCII >)**

Abort the [get an attribute](#) algorithm. The attribute's name is the value of *attribute name* and its value is the value of *attribute value*.

↪ **If it is in the range 0x41 (ASCII A) to 0x5A (ASCII Z)**

Append the Unicode character with code point *b*+0x20 to *attribute value* (where *b* is the value of the byte at *position*).

↪ **Anything else**

Append the Unicode character with the same code point as the value of the byte at *position* to *attribute value*.

12. Advance *position* to the next byte and return to the previous step.

For the sake of interoperability, user agents should not use a pre-scan algorithm that returns different results than the one described above. (But, if you do, please at least let us know, so that we can improve this algorithm and benefit everyone...)

8.2.2.3. Character encodings

User agents must support the encodings defined in the WHATWG Encoding standard, including, but not limited to,

UTF-8, ISO-8859-2, ISO-8859-8, windows-1250, windows-1251, windows-1252, windows-1254, windows-1256, windows-1257, gb18030, Big5, ISO-2022-JP, Shift_JIS, EUC-KR, UTF-16BE, UTF-16LE, and x-user-defined. User agents must not support other encodings.

NOTE:

The above prohibits supporting, for example, CESU-8, UTF-7, BOCU-1, SCSU, EBCDIC, and UTF-32. This specification does not make any attempt to support prohibited encodings in its algorithms; support and use of prohibited encodings would thus lead to unexpected behavior. [\[CESU8\]](#) [\[RFC2152\]](#) [\[BOCU1\]](#) [\[SCSU\]](#)

§ 8.2.2.4. Changing the encoding while parsing

When the parser requires the user agent to **change the encoding**, it must run the following steps. This might happen if the [encoding sniffing algorithm](#) described above failed to find a character encoding, or if it found a character encoding that was not the actual encoding of the file.

1. If the encoding that is already being used to interpret the input stream is a [UTF-16 encoding](#), then set the [confidence](#) to *certain* and abort these steps. The new encoding is ignored; if it was anything but the same encoding, then it would be clearly incorrect.
2. If the new encoding is a [UTF-16 encoding](#), then change it to [UTF-8](#).
3. If the new encoding is the [x-user-defined](#) encoding, then change it to [Windows-1252](#). [\[ENCODING\]](#)
4. If the new encoding is identical or equivalent to the encoding that is already being used to interpret the input stream, then set the [confidence](#) to *certain* and abort these steps. This happens when the encoding information found in the file matches what the [encoding sniffing algorithm](#) determined to be the encoding, and in the second pass through the parser if the first pass found that the encoding sniffing algorithm described in the earlier section failed to find the right encoding.
5. If all the bytes up to the last byte converted by the current decoder have the same Unicode interpretations in both the current encoding and the new encoding, and if the user agent supports changing the converter on the fly, then the user agent may change to the new converter for the encoding on the fly. Set the [document's character encoding](#) and the encoding used to convert the input stream to the new encoding, set the [confidence](#) to *certain*, and abort these steps.
6. Otherwise, [navigate](#) to the document again, with [replacement enabled](#), and using the same [source browsing context](#), but this time skip the [encoding sniffing algorithm](#) and instead just set the encoding to the new encoding and the [confidence](#) to *certain*. Whenever possible, this should be done without actually contacting the network layer (the bytes should be re-parsed from memory), even if, e.g., the document is marked as not being cacheable. If this is not possible and contacting the network layer would involve repeating a request that uses a method other than GET, then instead set the [confidence](#) to *certain* and ignore the new encoding. The resource will be misinterpreted. User agents may notify the user of the situation, to aid in application development.

NOTE:

This algorithm is only invoked when a new encoding is found declared on a [<meta>](#) element.

§ 8.2.2.5. Preprocessing the input stream

The **input stream** consists of the characters pushed into it as the [input byte stream](#) is decoded or from the various APIs that directly manipulate the input stream.

Any occurrences of any characters in the ranges U+0001 to U+0008, U+000E to U+001F, U+007F to U+009F, U+FDD0 to U+FDEF, and characters U+000B, U+FFFE, U+FFFF, U+1FFFE, U+1FFFF, U+2FFFE, U+2FFFF, U+3FFFE, U+3FFFF, U+4FFFE, U+4FFFF, U+5FFFE, U+5FFFF, U+6FFFE, U+6FFFF, U+7FFFE, U+7FFFF, U+8FFFE, U+8FFFF, U+9FFFE, U+9FFFF, U+AFFFE, U+AFFFF, U+BFFFE, U+BFFFF, U+CFFFE, U+CFFFF, U+DFFFE, U+DFFFF, U+EFFFE, U+EFFFF, U+FFFFE, U+FFFFF, U+10FFFE, and U+10FFFF are [parse errors](#). These are all [control characters](#) or permanently undefined Unicode characters (noncharacters).

Any [character](#) that is not a [Unicode character](#), i.e., any isolated surrogate, is a [parse error](#). (These can only find their way into the input stream via script APIs such as [document.write\(\)](#).)

U+000D CARRIAGE RETURN (CR) characters and U+000A LINE FEED (LF) characters are treated specially. Any LF character that immediately follows a CR character must be ignored, and all CR characters must then be converted to LF characters. Thus, newlines in HTML DOMs are represented by LF characters, and there are never any CR characters in the

input to the [tokenization](#) stage.

The **next input character** is the first character in the [input stream](#) that has not yet been **consumed** or explicitly ignored by the requirements in this section. Initially, the *next input character* is the first character in the input. The **current input character** is the last character to have been *consumed*.

The **insertion point** is the position (just before a character or just before the end of the input stream) where content inserted using `document.write()` is actually inserted. The insertion point is relative to the position of the character immediately after it, it is not an absolute offset into the input stream. Initially, the insertion point is undefined.

The "EOF" character in the tables below is a conceptual character representing the end of the [input stream](#). If the parser is a [script-created parser](#), then the end of the [input stream](#) is reached when an **explicit "EOF" character** (inserted by the `document.close()` method) is consumed. Otherwise, the "EOF" character is not a real character in the stream, but rather the lack of any further characters.

NOTE:

The handling of U+0000 NULL characters varies based on where the characters are found. In general, they are ignored except where doing so could plausibly introduce an attack vector. This handling is, by necessity, spread across both the tokenization stage and the tree construction stage.

§ 8.2.3. Parse state

§ 8.2.3.1. The insertion mode

The **insertion mode** is a state variable that controls the primary operation of the tree construction stage.

Initially, the [insertion mode](#) is "[initial](#)". It can change to "[before html](#)", "[before head](#)", "[in head](#)", "[in head noscript](#)", "[after head](#)", "[in body](#)", "[text](#)", "[in table](#)", "[in table text](#)", "[in caption](#)", "[in column group](#)", "[in table body](#)", "[in row](#)", "[in cell](#)", "[in select](#)", "[in select in table](#)", "[in template](#)", "[after body](#)", "[in frameset](#)", "[after frameset](#)", "[after after body](#)", and "[after after frameset](#)" during the course of the parsing, as described in the tree construction stage. The insertion mode affects how tokens are processed and whether CDATA sections are supported.

Several of these modes, namely "[in head](#)", "[in body](#)", "[in table](#)", and "[in select](#)", are special, in that the other modes defer to them at various times. When the algorithm below says that the user agent is to do something "**using the rules for the *m* insertion mode**", where *m* is one of these modes, the user agent must use the rules described under the *m* [insertion mode](#)'s section, but must leave the [insertion mode](#) unchanged unless the rules in *m* themselves switch the [insertion mode](#) to a new value.

When the insertion mode is switched to "[text](#)" or "[in table text](#)", the **original insertion mode** is also set. This is the insertion mode to which the tree construction stage will return.

Similarly, to parse nested `<template>` elements, a **stack of template insertion modes** is used. It is initially empty. The **current template insertion mode** is the insertion mode that was most recently added to the [stack of template insertion modes](#). The algorithms in the sections below will *push* insertion modes onto this stack, meaning that the specified insertion mode is to be added to the stack, and *pop* insertion modes from the stack, which means that the most recently added insertion mode must be removed from the stack.



When the steps below require the user agent to **reset the insertion mode appropriately**, it means the user agent must follow these steps:

1. Let *last* be false.
2. Let *node* be the last node in the [stack of open elements](#).
3. *Loop*: If *node* is the first node in the stack of open elements, then set *last* to true, and, if the parser was originally created as part of the [HTML fragment parsing algorithm](#) (*fragment case*), set *node* to the [context](#) element passed to that algorithm.
4. If *node* is a `<select>` element, run these substeps:

1. If *last* is true, jump to the step below labeled *done*.
2. Let *ancestor* be *node*.
3. *Loop*: If *ancestor* is the first node in the [stack of open elements](#), jump to the step below labeled *done*.
4. Let *ancestor* be the node before *ancestor* in the [stack of open elements](#).
5. If *ancestor* is a template node, jump to the step below labeled *done*.
6. If *ancestor* is a table node, switch the [insertion mode](#) to "[in select in table](#)" and abort these steps.
7. Jump back to the step labeled *loop*.
8. *Done*: Switch the [insertion mode](#) to "[in select](#)" and abort these steps.
5. If *node* is a `<td>` or `<th>` element and *last* is false, then switch the [insertion mode](#) to "[in cell](#)" and abort these steps.
6. If *node* is a `<tr>` element, then switch the [insertion mode](#) to "[in row](#)" and abort these steps.
7. If *node* is a `<tbody>`, `<thead>`, or `<tfoot>` element, then switch the [insertion mode](#) to "[in table body](#)" and abort these steps.
8. If *node* is a `<caption>` element, then switch the [insertion mode](#) to "[in caption](#)" and abort these steps.
9. If *node* is a `<colgroup>` element, then switch the [insertion mode](#) to "[in column group](#)" and abort these steps.
10. If *node* is a `<table>` element, then switch the [insertion mode](#) to "[in table](#)" and abort these steps.
11. If *node* is a `<template>` element, then switch the [insertion mode](#) to the [current template insertion mode](#) and abort these steps.
12. If *node* is a `<head>` element and *last* is false, then switch the [insertion mode](#) to "[in head](#)" and abort these steps.
13. If *node* is a `<body>` element, then switch the [insertion mode](#) to "[in body](#)" and abort these steps.
14. If *node* is a `<frameset>` element, then switch the [insertion mode](#) to "[in frameset](#)" and abort these steps. ([fragment case](#))
15. If *node* is an `<html>` element, run these substeps:
 1. If the [head element pointer](#) is null, switch the [insertion mode](#) to "[before head](#)" and abort these steps. ([fragment case](#))
 2. Otherwise, the [head element pointer](#) is not null, switch the [insertion mode](#) to "[after head](#)" and abort these steps.
16. If *last* is true, then switch the [insertion mode](#) to "[in body](#)" and abort these steps. ([fragment case](#))
17. Let *node* now be the node before *node* in the [stack of open elements](#).
18. Return to the step labeled *loop*.

§ 8.2.3.2. The stack of open elements

Initially, the **stack of open elements** is empty. The stack grows downwards; the topmost node on the stack is the first one added to the stack, and the bottommost node of the stack is the most recently added node in the stack (notwithstanding when the stack is manipulated in a random access fashion as part of [the handling for misnested tags](#)).

NOTE:

The "[before html](#)" [insertion mode](#) creates the `html` root element node, which is then added to the stack.

NOTE:

In the [fragment case](#), the [stack of open elements](#) is initialized to contain an `<html>` element that is created as part of [that algorithm](#). (The [fragment case](#) skips the "[before html](#)" [insertion mode](#).)

The `html` node, however it is created, is the topmost node of the stack. It only gets popped off the stack when the parser [fin-](#)

ishes.

The **current node** is the bottommost node in this [stack of open elements](#).

The **adjusted current node** is the [context](#) element if the parser was created by the [HTML fragment parsing algorithm](#) and the [stack of open elements](#) has only one element in it ([fragment case](#)); otherwise, the **adjusted current node** is the [current node](#).

Elements in the [stack of open elements](#) fall into the following categories:

Special

The following elements have varying levels of special parsing rules: HTML's [<address>](#), [<applet>](#), [<area>](#), [<article>](#), [<aside>](#), [<base>](#), [<basefont>](#), [<bgsound>](#), [<blockquote>](#), [<body>](#), [
](#), [<button>](#), [<caption>](#), [<center>](#), [<col>](#), [<colgroup>](#), [<dd>](#), [<details>](#), [<dir>](#), [<div>](#), [<dl>](#), [<dt>](#), [<embed>](#), [<fieldset>](#), [<figcaption>](#), [<figure>](#), [<footer>](#), [<form>](#), [<frame>](#), [<frameset>](#), [<h1>](#), [<h2>](#), [<h3>](#), [<h4>](#), [<h5>](#), [<h6>](#), [<head>](#), [<header>](#), [<hr>](#), [<html>](#), [<iframe>](#), [](#), [<input>](#), [](#), [<link>](#), [<listing>](#), [<main>](#), [<marquee>](#), [<menu>](#), [<menuitem>](#), [<meta>](#), [<nav>](#), [<noembed>](#), [<noframes>](#), [<noscript>](#), [<object>](#), [](#), [<p>](#), [<param>](#), [<plaintext>](#), [<pre>](#), [<script>](#), [<section>](#), [<select>](#), [<source>](#), [<style>](#), [<summary>](#), [<table>](#), [<tbody>](#), [<td>](#), [<template>](#), [<textarea>](#), [<tfoot>](#), [<th>](#), [<thead>](#), [<title>](#), [<tr>](#), [<track>](#), [](#), [<wbr>](#), and [<xmp>](#); MathML's [<mi>](#), [<mo>](#), [<mn>](#), [<ms>](#), [<mtext>](#), and [<annotation-xml>](#); and SVG's [<foreignObject>](#), [<desc>](#), and [<title>](#).

Formatting

The following HTML elements are those that end up in the [list of active formatting elements](#): [<a>](#), [](#), [<big>](#), [<code>](#), [](#), [](#), [<i>](#), [<noabr>](#), [<s>](#), [<small>](#), [<strike>](#), [](#), [<tt>](#), and [<u>](#).

Ordinary

All other elements found while parsing an HTML document.

The [stack of open elements](#) is said to **have an element target node in a specific scope** consisting of a list of element types *list* when the following algorithm terminates in a match state:

1. Initialize *node* to be the [current node](#) (the bottommost node of the stack).
2. If *node* is the target node, terminate in a match state.
3. Otherwise, if *node* is one of the element types in *list*, terminate in a failure state.
4. Otherwise, set *node* to the previous entry in the [stack of open elements](#) and return to step 2. (This will never fail, since the loop will always terminate in the previous step if the top of the stack — an [<html>](#) element — is reached.)

The [stack of open elements](#) is said to **have a particular element in scope** when it [has that element in the specific scope](#) consisting of the following element types:

- [<applet>](#) in the [HTML namespace](#)
- [<caption>](#) in the [HTML namespace](#)
- [<html>](#) in the [HTML namespace](#)
- [<table>](#) in the [HTML namespace](#)
- [<td>](#) in the [HTML namespace](#)
- [<th>](#) in the [HTML namespace](#)
- [<marquee>](#) in the [HTML namespace](#)
- [<object>](#) in the [HTML namespace](#)
- [<template>](#) in the [HTML namespace](#)
- [<mi>](#) in the [MathML namespace](#)
- [<mo>](#) in the [MathML namespace](#)
- [<mn>](#) in the [MathML namespace](#)
- [<ms>](#) in the [MathML namespace](#)
- [<mtext>](#) in the [MathML namespace](#)
- [<annotation-xml>](#) in the [MathML namespace](#)
- [<foreignObject>](#) in the [SVG namespace](#)
- [<desc>](#) in the [SVG namespace](#)
- [<title>](#) in the [SVG namespace](#)

The [stack of open elements](#) is said to **have a particular element in list item scope** when it [has that element in the specific scope](#) consisting of the following element types:

- All the element types listed above for the [has an element in scope](#) algorithm.
- [](#) in the [HTML namespace](#)
- [](#) in the [HTML namespace](#)

The [stack of open elements](#) is said to **have a particular element in button scope** when it [has that element in the specific](#)

[scope](#) consisting of the following element types:

- All the element types listed above for the [has an element in scope](#) algorithm.
- `<button>` in the [HTML namespace](#)

The [stack of open elements](#) is said to **have a particular element in table scope** when it [has that element in the specific scope](#) consisting of the following element types:

- `<html>` in the [HTML namespace](#)
- `<table>` in the [HTML namespace](#)
- `<template>` in the [HTML namespace](#)

The [stack of open elements](#) is said to **have a particular element in select scope** when it [has that element in the specific scope](#) consisting of all element types *except* the following:

- `<optgroup>` in the [HTML namespace](#)
- `<option>` in the [HTML namespace](#)

Nothing happens if at any time any of the elements in the [stack of open elements](#) are moved to a new location in, or removed from, the Document tree. In particular, the stack is not changed in this situation. This can cause, amongst other strange effects, content to be appended to nodes that are no longer in the DOM.

NOTE:

In some cases (namely, when [closing misnested formatting elements](#)), the stack is manipulated in a random-access fashion.

8.2.3.3. The list of active formatting elements

Initially, the **list of active formatting elements** is empty. It is used to handle mis-nested [formatting element tags](#).

The list contains elements in the [formatting](#) category, and [markers](#). The **markers** are inserted when entering applet elements, buttons, `<object>` elements, marquees, table cells, and table captions, and are used to prevent formatting from "leaking" into `<applet>` elements, buttons, `<object>` elements, marquees, and tables.

In addition, each element in the [list of active formatting elements](#) is associated with the token for which it was created, so that further elements can be created for that token if necessary.

When the steps below require the user agent to **push onto the list of active formatting elements** an element *element*, the user agent must perform the following steps:

1. If there are already three elements in the [list of active formatting elements](#) after the last [marker](#), if any, or anywhere in the list if there are no [markers](#), that have the same tag name, namespace, and attributes as *element*, then remove the earliest such element from the [list of active formatting elements](#). For these purposes, the attributes must be compared as they were when the elements were created by the parser; two elements have the same attributes if all their parsed attributes can be paired such that the two attributes in each pair have identical names, namespaces, and values (the order of the attributes does not matter).

NOTE:

This is the Noah's Ark clause. But with three per family instead of two.

2. Add *element* to the [list of active formatting elements](#).

When the steps below require the user agent to **reconstruct the active formatting elements**, the user agent must perform the following steps:

1. If there are no entries in the [list of active formatting elements](#), then there is nothing to reconstruct; stop this algorithm.
2. If the last (most recently added) entry in the [list of active formatting elements](#) is a [marker](#), or if it is an element that is in the [stack of open elements](#), then there is nothing to reconstruct; stop this algorithm.
3. Let *entry* be the last (most recently added) element in the [list of active formatting elements](#).
4. *Rewind*: If there are no entries before *entry* in the [list of active formatting elements](#), then jump to the step labeled *create*.

5. Let *entry* be the entry one earlier than *entry* in the [list of active formatting elements](#).
6. If *entry* is neither a [marker](#) nor an element that is also in the [stack of open elements](#), go to the step labeled *rewind*.
7. *Advance*: Let *entry* be the element one later than *entry* in the [list of active formatting elements](#).
8. *Create*: [Insert an HTML element](#) for the token for which the element *entry* was created, to obtain *new element*.
9. Replace the entry for *entry* in the list with an entry for *new element*.
10. If the entry for *new element* in the [list of active formatting elements](#) is not the last entry in the list, return to the step labeled *advance*.

This has the effect of reopening all the formatting elements that were opened in the current body, cell, or caption (whichever is youngest) that haven't been explicitly closed.

NOTE:

The way this specification is written, the [list of active formatting elements](#) always consists of elements in chronological order with the least recently added element first and the most recently added element last (except for while steps 7 to 10 of the above algorithm are being executed, of course).

When the steps below require the user agent to **clear the list of active formatting elements up to the last marker**, the user agent must perform the following steps:

1. Let *entry* be the last (most recently added) entry in the [list of active formatting elements](#).
2. Remove *entry* from the [list of active formatting elements](#).
3. If *entry* was a [marker](#), then stop the algorithm at this point. The list has been cleared up to the last [marker](#).
4. Go to step 1.

§ 8.2.3.4. The element pointers

Initially, the **head element pointer** and the **form element pointer** are both null.

Once a [<head>](#) element has been parsed (whether implicitly or explicitly) the [head element pointer](#) gets set to point to this node.

The [form element pointer](#) points to the last [<form>](#) element that was opened and whose end tag has not yet been seen. It is used to make form controls associate with forms in the face of dramatically bad markup, for historical reasons. It is ignored inside [<template>](#) elements.

§ 8.2.3.5. Other parsing state flags

The **scripting flag** is set to "enabled" if [scripting was enabled](#) for the Document with which the parser is associated when the parser was created, and "disabled" otherwise.

NOTE:

The [scripting flag](#) can be enabled even when the parser was originally created for the [HTML fragment parsing algorithm](#), even though `script` elements don't execute in that case.

The **frameset-ok flag** is set to "ok" when the parser is created. It is set to "not ok" after certain tokens are seen.

§ 8.2.4. Tokenization

Implementations must act as if they used the following state machine to tokenize HTML. The state machine must start in the [§8.2.4.1 Data state](#). Most states consume a single character, which may have various side-effects, and either switches the state machine to a new state to *reconsume* the same character, or switches it to a new state to consume the next character, or stays in the same state to consume the next character. Some states have more complicated behavior and can consume several

characters before switching to another state. In some cases, the tokenizer state is also changed by the tree construction stage.

The exact behavior of certain states depends on the [insertion mode](#) and the [stack of open elements](#). Certain states also use a **temporary buffer** to track progress.

The output of the tokenization step is a series of zero or more of the following tokens: DOCTYPE, start tag, end tag, comment, character, end-of-file. DOCTYPE tokens have a name, a public identifier, a system identifier, and a *force-quirks flag*. When a DOCTYPE token is created, its name, public identifier, and system identifier must be marked as missing (which is a distinct state from the empty string), and the *force-quirks flag* must be set to *off* (its other state is *on*). Start and end tag tokens have a tag name, a **self-closing flag**, and a list of attributes, each of which has a name and a value. When a start or end tag token is created, its *self-closing flag* must be unset (its other state is that it be set), and its attributes list must be empty. Comment and character tokens have data.

When a token is emitted, it must immediately be handled by the tree construction stage. The tree construction stage can affect the state of the tokenization stage, and can insert additional characters into the stream. (For example, the `<script>` element can result in scripts executing and using the [dynamic markup insertion](#) APIs to insert characters into the stream being tokenized.)

NOTE:

Creating a token and emitting it are distinct actions. It is possible for a token to be created but implicitly abandoned (never emitted), e.g., if the file ends unexpectedly while processing the characters that are being parsed into a start tag token.

When a start tag token is emitted with its *self-closing flag* set, if the flag is not **acknowledged** when it is processed by the tree construction stage, that is a [parse error](#).

When an end tag token is emitted with attributes, that is a [parse error](#).

When an end tag token is emitted with its *self-closing flag* set, that is a [parse error](#).

An **appropriate end tag token** is an end tag token whose tag name matches the tag name of the last start tag to have been emitted from this tokenizer, if any. If no start tag has been emitted from this tokenizer, then no end tag token is appropriate.

Before each step of the tokenizer, the user agent must first check the [parser pause flag](#). If it is true, then the tokenizer must abort the processing of any nested invocations of the tokenizer, yielding control back to the caller.

The tokenizer state machine consists of the states defined in the following subsections.

8.2.4.1. Data state

Consume the [next input character](#):

↪ U+0026 AMPERSAND (&)

Switch to the [§8.2.4.2 Character reference in data state](#).

↪ U+003C LESS-THAN SIGN (<)

Switch to the [§8.2.4.8 Tag open state](#).

↪ U+0000 NULL

[parse error](#). Emit the [current input character](#) as a character token.

↪ EOF

Emit an end-of-file token.

↪ Anything else

Emit the [current input character](#) as a character token.

8.2.4.2. Character reference in data state

Switch to the [§8.2.4.1 Data state](#).

Attempt to [consume a character reference](#), with no [additional allowed character](#).

If nothing is returned, emit a U+0026 AMPERSAND character (&) token.

Otherwise, emit the character tokens that were returned.

§ 8.2.4.3. RCDATA state

Consume the [next input character](#):

↪ **U+0026 AMPERSAND (&)**

Switch to the [§8.2.4.4 Character reference in RCDATA state](#).

↪ **U+003C LESS-THAN SIGN (<)**

Switch to the [§8.2.4.11 RCDATA less-than sign state](#).

↪ **U+0000 NULL**

[parse error](#). Emit a U+FFFD REPLACEMENT CHARACTER character token.

↪ **EOF**

Emit an end-of-file token.

↪ **Anything else**

Emit the [current input character](#) as a character token.

§ 8.2.4.4. Character reference in RCDATA state

Switch to the [§8.2.4.3 RCDATA state](#).

Attempt to [consume a character reference](#), with no [additional allowed character](#).

If nothing is returned, emit a U+0026 AMPERSAND character (&) token.

Otherwise, emit the character tokens that were returned.

§ 8.2.4.5. RAWTEXT state

Consume the [next input character](#):

↪ **U+003C LESS-THAN SIGN (<)**

Switch to the [§8.2.4.14 RAWTEXT less-than sign state](#).

↪ **U+0000 NULL**

[parse error](#). Emit a U+FFFD REPLACEMENT CHARACTER character token.

↪ **EOF**

Emit an end-of-file token.

↪ **Anything else**

Emit the [current input character](#) as a character token.

§ 8.2.4.6. Script data state

Consume the [next input character](#):

↪ **U+003C LESS-THAN SIGN (<)**

Switch to the [§8.2.4.17 Script data less-than sign state](#).

↪ **U+0000 NULL**

[parse error](#). Emit a U+FFFD REPLACEMENT CHARACTER character token.

↪ EOF

Emit an end-of-file token.

↪ Anything else

Emit the [current input character](#) as a character token.

§ 8.2.4.7. *PLAINTEXT state*

Consume the [next input character](#):

↪ U+0000 NULL

[parse error](#). Emit a U+FFFD REPLACEMENT CHARACTER character token.

↪ EOF

Emit an end-of-file token.

↪ Anything else

Emit the [current input character](#) as a character token.

§ 8.2.4.8. *Tag open state*

Consume the [next input character](#):

↪ U+0021 EXCLAMATION MARK (!)

Switch to the [§8.2.4.45 Markup declaration open state](#).

↪ U+002F SOLIDUS (/)

Switch to the [§8.2.4.9 End tag open state](#).

↪ [Uppercase ASCII letter](#)

Create a new start tag token, set its tag name to the lowercase version of the [current input character](#) (add 0x0020 to the character's code point), then switch to the [§8.2.4.10 Tag name state](#). (Don't emit the token yet; further details will be filled in before it is emitted.)

↪ [Lowercase ASCII letter](#)

Create a new start tag token, set its tag name to the [current input character](#), then switch to the [§8.2.4.10 Tag name state](#). (Don't emit the token yet; further details will be filled in before it is emitted.)

↪ U+003F QUESTION MARK (?)

[parse error](#). Switch to the [§8.2.4.44 Bogus comment state](#).

↪ Anything else

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Emit a U+003C LESS-THAN SIGN character token. Reconsume the [current input character](#).

§ 8.2.4.9. *End tag open state*

Consume the [next input character](#):

↪ [Uppercase ASCII letter](#)

Create a new end tag token, set its tag name to the lowercase version of the [current input character](#) (add 0x0020 to the character's code point), then switch to the [§8.2.4.10 Tag name state](#). (Don't emit the token yet; further details will be filled in before it is emitted.)

↪ [Lowercase ASCII letter](#)

Create a new end tag token, set its tag name to the [current input character](#), then switch to the [§8.2.4.10 Tag name state](#). (Don't emit the token yet; further details will be filled in before it is emitted.)

↪ U+003E GREATER-THAN SIGN (>)

[parse error](#). Switch to the [§8.2.4.1 Data state](#).

↪ EOF

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Emit a U+003C LESS-THAN SIGN character token and a U+002F SOLIDUS character token. Reconsume the EOF character.

↪ Anything else

[parse error](#). Switch to the [§8.2.4.44 Bogus comment state](#).

§ 8.2.4.10. Tag name state

Consume the [next input character](#):

↪ U+0009 CHARACTER TABULATION (tab)

↪ U+000A LINE FEED (LF)

↪ U+000C FORM FEED (FF)

↪ U+0020 SPACE

Switch to the [§8.2.4.34 Before attribute name state](#).

↪ U+002F SOLIDUS (/)

Switch to the [§8.2.4.43 Self-closing start tag state](#).

↪ U+003E GREATER-THAN SIGN (>)

Switch to the [§8.2.4.1 Data state](#). Emit the current tag token.

↪ [Uppercase ASCII letter](#)

Append the lowercase version of the [current input character](#) (add 0x0020 to the character's code point) to the current tag token's tag name.

↪ U+0000 NULL

[parse error](#). Append a U+FFFD REPLACEMENT CHARACTER character to the current tag token's tag name.

↪ EOF

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Reconsume the EOF character.

↪ Anything else

Append the [current input character](#) to the current tag token's tag name.

§ 8.2.4.11. RCDATA less-than sign state

Consume the [next input character](#):

↪ U+002F SOLIDUS (/)

Set the [temporary buffer](#) to the empty string. Switch to the [§8.2.4.12 RCDATA end tag open state](#).

↪ Anything else

Switch to the [§8.2.4.3 RCDATA state](#). Emit a U+003C LESS-THAN SIGN character token. Reconsume the [current input character](#).

§ 8.2.4.12. RCDATA end tag open state

Consume the [next input character](#):

↪ [Uppercase ASCII letter](#)

Create a new end tag token, and set its tag name to the lowercase version of the [current input character](#) (add 0x0020 to the character's code point). Append the [current input character](#) to the [temporary buffer](#). Finally, switch to the [§8.2.4.13 RCDATA end tag name state](#). (Don't emit the token yet; further details will be filled in before it is emitted.)

↪ [Lowercase ASCII letter](#)

Create a new end tag token, and set its tag name to the [current input character](#). Append the [current input character](#) to the *temporary buffer*. Finally, switch to the [§8.2.4.13 RCDATA end tag name state](#). (Don't emit the token yet; further details will be filled in before it is emitted.)

↪ **Anything else**

Switch to the [§8.2.4.3 RCDATA state](#). Emit a U+003C LESS-THAN SIGN character token and a U+002F SOLIDUS character token. Reconsume the [current input character](#).

§ 8.2.4.13. *RCDATA end tag name state*

Consume the [next input character](#):

↪ **U+0009 CHARACTER TABULATION (tab)**

↪ **U+000A LINE FEED (LF)**

↪ **U+000C FORM FEED (FF)**

↪ **U+0020 SPACE**

If the current end tag token is an [appropriate end tag token](#), then switch to the [§8.2.4.34 Before attribute name state](#). Otherwise, treat it as per the "anything else" entry below.

↪ **U+002F SOLIDUS (/)**

If the current end tag token is an [appropriate end tag token](#), then switch to the [§8.2.4.43 Self-closing start tag state](#). Otherwise, treat it as per the "anything else" entry below.

↪ **U+003E GREATER-THAN SIGN (>)**

If the current end tag token is an [appropriate end tag token](#), then switch to the [§8.2.4.1 Data state](#) and emit the current tag token. Otherwise, treat it as per the "anything else" entry below.

↪ [Uppercase ASCII letter](#)

Append the lowercase version of the [current input character](#) (add 0x0020 to the character's code point) to the current tag token's tag name. Append the [current input character](#) to the *temporary buffer*.

↪ [Lowercase ASCII letter](#)

Append the [current input character](#) to the current tag token's tag name. Append the [current input character](#) to the *temporary buffer*.

↪ **Anything else**

Switch to the [§8.2.4.3 RCDATA state](#). Emit a U+003C LESS-THAN SIGN character token, a U+002F SOLIDUS character token, and a character token for each of the characters in the *temporary buffer* (in the order they were added to the buffer). Reconsume the [current input character](#).

§ 8.2.4.14. *RAWTEXT less-than sign state*

Consume the [next input character](#):

↪ **U+002F SOLIDUS (/)**

Set the *temporary buffer* to the empty string. Switch to the [§8.2.4.15 RAWTEXT end tag open state](#).

↪ **Anything else**

Switch to the [§8.2.4.5 RAWTEXT state](#). Emit a U+003C LESS-THAN SIGN character token. Reconsume the [current input character](#).

§ 8.2.4.15. *RAWTEXT end tag open state*

Consume the [next input character](#):

↪ [Uppercase ASCII letter](#)

Create a new end tag token, and set its tag name to the lowercase version of the [current input character](#) (add

0x0020 to the character's code point). Append the [current input character](#) to the *temporary buffer*. Finally, switch to the [§8.2.4.16 RAWTEXT end tag name state](#). (Don't emit the token yet; further details will be filled in before it is emitted.)

↪ **Lowercase ASCII letter**

Create a new end tag token, and set its tag name to the [current input character](#). Append the [current input character](#) to the *temporary buffer*. Finally, switch to the [§8.2.4.16 RAWTEXT end tag name state](#). (Don't emit the token yet; further details will be filled in before it is emitted.)

↪ **Anything else**

Switch to the [§8.2.4.5 RAWTEXT state](#). Emit a U+003C LESS-THAN SIGN character token and a U+002F SOLIDUS character token. Reconsume the [current input character](#).

§ 8.2.4.16. RAWTEXT end tag name state

Consume the [next input character](#):

↪ **U+0009 CHARACTER TABULATION (tab)**

↪ **U+000A LINE FEED (LF)**

↪ **U+000C FORM FEED (FF)**

↪ **U+0020 SPACE**

If the current end tag token is an [appropriate end tag token](#), then switch to the [§8.2.4.34 Before attribute name state](#). Otherwise, treat it as per the "anything else" entry below.

↪ **U+002F SOLIDUS (/)**

If the current end tag token is an [appropriate end tag token](#), then switch to the [§8.2.4.43 Self-closing start tag state](#). Otherwise, treat it as per the "anything else" entry below.

↪ **U+003E GREATER-THAN SIGN (>)**

If the current end tag token is an [appropriate end tag token](#), then switch to the [§8.2.4.1 Data state](#) and emit the current tag token. Otherwise, treat it as per the "anything else" entry below.

↪ **Uppercase ASCII letter**

Append the lowercase version of the [current input character](#) (add 0x0020 to the character's code point) to the current tag token's tag name. Append the [current input character](#) to the *temporary buffer*.

↪ **Lowercase ASCII letter**

Append the [current input character](#) to the current tag token's tag name. Append the [current input character](#) to the *temporary buffer*.

↪ **Anything else**

Switch to the [§8.2.4.5 RAWTEXT state](#). Emit a U+003C LESS-THAN SIGN character token, a U+002F SOLIDUS character token, and a character token for each of the characters in the *temporary buffer* (in the order they were added to the buffer). Reconsume the [current input character](#).

§ 8.2.4.17. Script data less-than sign state

Consume the [next input character](#):

↪ **U+002F SOLIDUS (/)**

Set the *temporary buffer* to the empty string. Switch to the [§8.2.4.18 Script data end tag open state](#).

↪ **U+0021 EXCLAMATION MARK (!)**

Switch to the [§8.2.4.20 Script data escape start state](#). Emit a U+003C LESS-THAN SIGN character token and a U+0021 EXCLAMATION MARK character token.

↪ **Anything else**

Switch to the [§8.2.4.6 Script data state](#). Emit a U+003C LESS-THAN SIGN character token. Reconsume the [current input character](#).

§ 8.2.4.18. Script data end tag open state

Consume the [next input character](#):

↪ [Uppercase ASCII letter](#)

Create a new end tag token, and set its tag name to the lowercase version of the [current input character](#) (add 0x0020 to the character's code point). Append the [current input character](#) to the *temporary buffer*. Finally, switch to the [§8.2.4.19 Script data end tag name state](#). (Don't emit the token yet; further details will be filled in before it is emitted.)

↪ [Lowercase ASCII letter](#)

Create a new end tag token, and set its tag name to the [current input character](#). Append the [current input character](#) to the *temporary buffer*. Finally, switch to the [§8.2.4.19 Script data end tag name state](#). (Don't emit the token yet; further details will be filled in before it is emitted.)

↪ [Anything else](#)

Switch to the [§8.2.4.6 Script data state](#). Emit a U+003C LESS-THAN SIGN character token and a U+002F SOLIDUS character token. Reconsume the [current input character](#).

§ 8.2.4.19. Script data end tag name state

Consume the [next input character](#):

↪ [U+0009 CHARACTER TABULATION \(tab\)](#)

↪ [U+000A LINE FEED \(LF\)](#)

↪ [U+000C FORM FEED \(FF\)](#)

↪ [U+0020 SPACE](#)

If the current end tag token is an [appropriate end tag token](#), then switch to the [§8.2.4.34 Before attribute name state](#). Otherwise, treat it as per the "anything else" entry below.

↪ [U+002F SOLIDUS \(/\)](#)

If the current end tag token is an [appropriate end tag token](#), then switch to the [§8.2.4.43 Self-closing start tag state](#). Otherwise, treat it as per the "anything else" entry below.

↪ [U+003E GREATER-THAN SIGN \(>\)](#)

If the current end tag token is an [appropriate end tag token](#), then switch to the [§8.2.4.1 Data state](#) and emit the current tag token. Otherwise, treat it as per the "anything else" entry below.

↪ [Uppercase ASCII letter](#)

Append the lowercase version of the [current input character](#) (add 0x0020 to the character's code point) to the current tag token's tag name. Append the [current input character](#) to the *temporary buffer*.

↪ [Lowercase ASCII letter](#)

Append the [current input character](#) to the current tag token's tag name. Append the [current input character](#) to the *temporary buffer*.

↪ [Anything else](#)

Switch to the [§8.2.4.6 Script data state](#). Emit a U+003C LESS-THAN SIGN character token, a U+002F SOLIDUS character token, and a character token for each of the characters in the *temporary buffer* (in the order they were added to the buffer). Reconsume the [current input character](#).

§ 8.2.4.20. Script data escape start state

Consume the [next input character](#):

↪ [U+002D HYPHEN-MINUS \(-\)](#)

Switch to the [§8.2.4.21 Script data escape start dash state](#). Emit a U+002D HYPHEN-MINUS character token.

↪ [Anything else](#)

Switch to the [§8.2.4.6 Script data state](#). Reconsume the [current input character](#).

§ 8.2.4.21. Script data escape start dash state

Consume the [next input character](#):

↪ U+002D HYPHEN-MINUS (-)

Switch to the [§8.2.4.24 Script data escaped dash dash state](#). Emit a U+002D HYPHEN-MINUS character token.

↪ Anything else

Switch to the [§8.2.4.6 Script data state](#). Reconsume the [current input character](#).

§ 8.2.4.22. Script data escaped state

Consume the [next input character](#):

↪ U+002D HYPHEN-MINUS (-)

Switch to the [§8.2.4.23 Script data escaped dash state](#). Emit a U+002D HYPHEN-MINUS character token.

↪ U+003C LESS-THAN SIGN (<)

Switch to the [§8.2.4.25 Script data escaped less-than sign state](#).

↪ U+0000 NULL

[parse error](#). Emit a U+FFFD REPLACEMENT CHARACTER character token.

↪ EOF

Switch to the [§8.2.4.1 Data state](#). [parse error](#). Reconsume the EOF character.

↪ Anything else

Emit the [current input character](#) as a character token.

§ 8.2.4.23. Script data escaped dash state

Consume the [next input character](#):

↪ U+002D HYPHEN-MINUS (-)

Switch to the [§8.2.4.24 Script data escaped dash dash state](#). Emit a U+002D HYPHEN-MINUS character token.

↪ U+003C LESS-THAN SIGN (<)

Switch to the [§8.2.4.25 Script data escaped less-than sign state](#).

↪ U+0000 NULL

[parse error](#). Switch to the [§8.2.4.22 Script data escaped state](#). Emit a U+FFFD REPLACEMENT CHARACTER character token.

↪ EOF

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Reconsume the EOF character.

↪ Anything else

Switch to the [§8.2.4.22 Script data escaped state](#). Emit the [current input character](#) as a character token.

§ 8.2.4.24. Script data escaped dash dash state

Consume the [next input character](#):

↪ U+002D HYPHEN-MINUS (-)

Emit a U+002D HYPHEN-MINUS character token.

↪ U+003C LESS-THAN SIGN (<)

Switch to the [§8.2.4.25 Script data escaped less-than sign state](#).

↪ **U+003E GREATER-THAN SIGN (>)**

Switch to the [§8.2.4.6 Script data state](#). Emit a U+003E GREATER-THAN SIGN character token.

↪ **U+0000 NULL**

[parse error](#). Switch to the [§8.2.4.22 Script data escaped state](#). Emit a U+FFFD REPLACEMENT CHARACTER character token.

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Reconsume the EOF character.

↪ **Anything else**

Switch to the [§8.2.4.22 Script data escaped state](#). Emit the [current input character](#) as a character token.

§ 8.2.4.25. *Script data escaped less-than sign state*

Consume the [next input character](#):

↪ **U+002F SOLIDUS (/)**

Set the [temporary buffer](#) to the empty string. Switch to the [§8.2.4.26 Script data escaped end tag open state](#).

↪ **Uppercase ASCII letter**

Set the [temporary buffer](#) to the empty string. Append the lowercase version of the [current input character](#) (add 0x0020 to the character's code point) to the [temporary buffer](#). Switch to the [§8.2.4.28 Script data double escape start state](#). Emit a U+003C LESS-THAN SIGN character token and the [current input character](#) as a character token.

↪ **Lowercase ASCII letter**

Set the [temporary buffer](#) to the empty string. Append the [current input character](#) to the [temporary buffer](#). Switch to the [§8.2.4.28 Script data double escape start state](#). Emit a U+003C LESS-THAN SIGN character token and the [current input character](#) as a character token.

↪ **Anything else**

Switch to the [§8.2.4.22 Script data escaped state](#). Emit a U+003C LESS-THAN SIGN character token. Reconsume the [current input character](#).

§ 8.2.4.26. *Script data escaped end tag open state*

Consume the [next input character](#):

↪ **Uppercase ASCII letter**

Create a new end tag token, and set its tag name to the lowercase version of the [current input character](#) (add 0x0020 to the character's code point). Append the [current input character](#) to the [temporary buffer](#). Finally, switch to the [§8.2.4.27 Script data escaped end tag name state](#). (Don't emit the token yet; further details will be filled in before it is emitted.)

↪ **Lowercase ASCII letter**

Create a new end tag token, and set its tag name to the [current input character](#). Append the [current input character](#) to the [temporary buffer](#). Finally, switch to the [§8.2.4.27 Script data escaped end tag name state](#). (Don't emit the token yet; further details will be filled in before it is emitted.)

↪ **Anything else**

Switch to the [§8.2.4.22 Script data escaped state](#). Emit a U+003C LESS-THAN SIGN character token and a U+002F SOLIDUS character token. Reconsume the [current input character](#).

§ 8.2.4.27. *Script data escaped end tag name state*

Consume the [next input character](#):

↪ **U+0009 CHARACTER TABULATION (tab)**

↪ **U+000A LINE FEED (LF)**

↪ **U+000C FORM FEED (FF)**

↪ **U+0020 SPACE**

If the current end tag token is an [appropriate end tag token](#), then switch to the [§8.2.4.34 Before attribute name state](#). Otherwise, treat it as per the "anything else" entry below.

↪ **U+002F SOLIDUS (/)**

If the current end tag token is an [appropriate end tag token](#), then switch to the [§8.2.4.43 Self-closing start tag state](#). Otherwise, treat it as per the "anything else" entry below.

↪ **U+003E GREATER-THAN SIGN (>)**

If the current end tag token is an [appropriate end tag token](#), then switch to the [§8.2.4.1 Data state](#) and emit the current tag token. Otherwise, treat it as per the "anything else" entry below.

↪ **Uppercase ASCII letter**

Append the lowercase version of the [current input character](#) (add 0x0020 to the character's code point) to the current tag token's tag name. Append the [current input character](#) to the *temporary buffer*.

↪ **Lowercase ASCII letter**

Append the [current input character](#) to the current tag token's tag name. Append the [current input character](#) to the *temporary buffer*.

↪ **Anything else**

Switch to the [§8.2.4.22 Script data escaped state](#). Emit a U+003C LESS-THAN SIGN character token, a U+002F SOLIDUS character token, and a character token for each of the characters in the *temporary buffer* (in the order they were added to the buffer). Reconsume the [current input character](#).

§ 8.2.4.28. *Script data double escape start state*

Consume the [next input character](#):

↪ **U+0009 CHARACTER TABULATION (tab)**

↪ **U+000A LINE FEED (LF)**

↪ **U+000C FORM FEED (FF)**

↪ **U+0020 SPACE**

↪ **U+002F SOLIDUS (/)**

↪ **U+003E GREATER-THAN SIGN (>)**

If the *temporary buffer* is the string "script", then switch to the [§8.2.4.29 Script data double escaped state](#). Otherwise, switch to the [§8.2.4.22 Script data escaped state](#). Emit the [current input character](#) as a character token.

↪ **Uppercase ASCII letter**

Append the lowercase version of the [current input character](#) (add 0x0020 to the character's code point) to the *temporary buffer*. Emit the [current input character](#) as a character token.

↪ **Lowercase ASCII letter**

Append the [current input character](#) to the *temporary buffer*. Emit the [current input character](#) as a character token.

↪ **Anything else**

Switch to the [§8.2.4.22 Script data escaped state](#). Reconsume the [current input character](#).

§ 8.2.4.29. *Script data double escaped state*

Consume the [next input character](#):

↪ **U+002D HYPHEN-MINUS (-)**

Switch to the [§8.2.4.30 Script data double escaped dash state](#). Emit a U+002D HYPHEN-MINUS character token.

↪ **U+003C LESS-THAN SIGN (<)**

Switch to the [§8.2.4.32 Script data double escaped less-than sign state](#). Emit a U+003C LESS-THAN SIGN character token.

↪ **U+0000 NULL**

[parse error](#). Emit a U+FFFD REPLACEMENT CHARACTER character token.

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Reconsume the EOF character.

↪ **Anything else**

Emit the [current input character](#) as a character token.

§ 8.2.4.30. *Script data double escaped dash state*

Consume the [next input character](#):

↪ **U+002D HYPHEN-MINUS (-)**

Switch to the [§8.2.4.31 Script data double escaped dash dash state](#). Emit a U+002D HYPHEN-MINUS character token.

↪ **U+003C LESS-THAN SIGN (<)**

Switch to the [§8.2.4.32 Script data double escaped less-than sign state](#). Emit a U+003C LESS-THAN SIGN character token.

↪ **U+0000 NULL**

[parse error](#). Switch to the [§8.2.4.29 Script data double escaped state](#). Emit a U+FFFD REPLACEMENT CHARACTER character token.

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Reconsume the EOF character.

↪ **Anything else**

Switch to the [§8.2.4.29 Script data double escaped state](#). Emit the [current input character](#) as a character token.

§ 8.2.4.31. *Script data double escaped dash dash state*

Consume the [next input character](#):

↪ **U+002D HYPHEN-MINUS (-)**

Emit a U+002D HYPHEN-MINUS character token.

↪ **U+003C LESS-THAN SIGN (<)**

Switch to the [§8.2.4.32 Script data double escaped less-than sign state](#). Emit a U+003C LESS-THAN SIGN character token.

↪ **U+003E GREATER-THAN SIGN (>)**

Switch to the [§8.2.4.6 Script data state](#). Emit a U+003E GREATER-THAN SIGN character token.

↪ **U+0000 NULL**

[parse error](#). Switch to the [§8.2.4.29 Script data double escaped state](#). Emit a U+FFFD REPLACEMENT CHARACTER character token.

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Reconsume the EOF character.

↪ **Anything else**

Switch to the [§8.2.4.29 Script data double escaped state](#). Emit the [current input character](#) as a character token.

§ 8.2.4.32. Script data double escaped less-than sign state

Consume the [next input character](#):

↪ U+002F SOLIDUS (/)

Set the *temporary buffer* to the empty string. Switch to the [§8.2.4.33 Script data double escape end state](#). Emit a U+002F SOLIDUS character token.

↪ Anything else

Switch to the [§8.2.4.29 Script data double escaped state](#). Reconsume the [current input character](#).

§ 8.2.4.33. Script data double escape end state

Consume the [next input character](#):

↪ U+0009 CHARACTER TABULATION (tab)

↪ U+000A LINE FEED (LF)

↪ U+000C FORM FEED (FF)

↪ U+0020 SPACE

↪ U+002F SOLIDUS (/)

↪ U+003E GREATER-THAN SIGN (>)

If the *temporary buffer* is the string "script", then switch to the [§8.2.4.22 Script data escaped state](#). Otherwise, switch to the [§8.2.4.29 Script data double escaped state](#). Emit the [current input character](#) as a character token.

↪ Uppercase ASCII letter

Append the lowercase version of the [current input character](#) (add 0x0020 to the character's code point) to the *temporary buffer*. Emit the [current input character](#) as a character token.

↪ Lowercase ASCII letter

Append the [current input character](#) to the *temporary buffer*. Emit the [current input character](#) as a character token.

↪ Anything else

Switch to the [§8.2.4.29 Script data double escaped state](#). Reconsume the [current input character](#).

§ 8.2.4.34. Before attribute name state

Consume the [next input character](#):

↪ U+0009 CHARACTER TABULATION (tab)

↪ U+000A LINE FEED (LF)

↪ U+000C FORM FEED (FF)

↪ U+0020 SPACE

Ignore the character.

↪ U+002F SOLIDUS (/)

Switch to the [§8.2.4.43 Self-closing start tag state](#).

↪ U+003E GREATER-THAN SIGN (>)

Switch to the [§8.2.4.1 Data state](#). Emit the current tag token.

↪ Uppercase ASCII letter

Start a new attribute in the current tag token. Set that attribute's name to the lowercase version of the [current input character](#) (add 0x0020 to the character's code point), and its value to the empty string. Switch to the [§8.2.4.35 Attribute name state](#).

↪ U+0000 NULL

[parse error](#). Start a new attribute in the current tag token. Set that attribute's name to a U+FFFD REPLACEMENT CHARACTER character, and its value to the empty string. Switch to the [§8.2.4.35 Attribute name state](#).

↪ **U+0022 QUOTATION MARK (")**

↪ **U+0027 APOSTROPHE (')**

↪ **U+003C LESS-THAN SIGN (<)**

↪ **U+003D EQUALS SIGN (=)**

[parse error](#). Treat it as per the "anything else" entry below.

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Reconsume the EOF character.

↪ **Anything else**

Start a new attribute in the current tag token. Set that attribute's name to the [current input character](#), and its value to the empty string. Switch to the [§8.2.4.35 Attribute name state](#).

§ 8.2.4.35. Attribute name state

Consume the [next input character](#):

↪ **U+0009 CHARACTER TABULATION (tab)**

↪ **U+000A LINE FEED (LF)**

↪ **U+000C FORM FEED (FF)**

↪ **U+0020 SPACE**

Switch to the [§8.2.4.36 After attribute name state](#).

↪ **U+002F SOLIDUS (/)**

Switch to the [§8.2.4.43 Self-closing start tag state](#).

↪ **U+003D EQUALS SIGN (=)**

Switch to the [§8.2.4.37 Before attribute value state](#).

↪ **U+003E GREATER-THAN SIGN (>)**

Switch to the [§8.2.4.1 Data state](#). Emit the current tag token.

↪ **Uppercase ASCII letter**

Append the lowercase version of the [current input character](#) (add 0x0020 to the character's code point) to the current attribute's name.

↪ **U+0000 NULL**

[parse error](#). Append a U+FFFD REPLACEMENT CHARACTER character to the current attribute's name.

↪ **U+0022 QUOTATION MARK (")**

↪ **U+0027 APOSTROPHE (')**

↪ **U+003C LESS-THAN SIGN (<)**

[parse error](#). Treat it as per the "anything else" entry below.

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Reconsume the EOF character.

↪ **Anything else**

Append the [current input character](#) to the current attribute's name.

When the user agent leaves the attribute name state (and before emitting the tag token, if appropriate), the complete attribute's name must be compared to the other attributes on the same token; if there is already an attribute on the token with the exact same name, then this is a [parse error](#) and the new attribute must be removed from the token.

NOTE:

If an attribute is so removed from a token, it, and the value that gets associated with it, if any, are never subsequently used by the parser, and are therefore effectively discarded. Removing the attribute in this way does not change its status as the "current attribute" for the purposes of the tokenizer, however.

§ 8.2.4.36. After attribute name state

Consume the [next input character](#):

↪ **U+0009 CHARACTER TABULATION (tab)**

↪ **U+000A LINE FEED (LF)**

↪ **U+000C FORM FEED (FF)**

↪ **U+0020 SPACE**

Ignore the character.

↪ **U+002F SOLIDUS (/)**

Switch to the [§8.2.4.43 Self-closing start tag state](#).

↪ **U+003D EQUALS SIGN (=)**

Switch to the [§8.2.4.37 Before attribute value state](#).

↪ **U+003E GREATER-THAN SIGN (>)**

Switch to the [§8.2.4.1 Data state](#). Emit the current tag token.

↪ **Uppercase ASCII letter**

Start a new attribute in the current tag token. Set that attribute's name to the lowercase version of the [current input character](#) (add 0x0020 to the character's code point), and its value to the empty string. Switch to the [§8.2.4.35 Attribute name state](#).

↪ **U+0000 NULL**

[parse error](#). Start a new attribute in the current tag token. Set that attribute's name to a U+FFFD REPLACEMENT CHARACTER character, and its value to the empty string. Switch to the [§8.2.4.35 Attribute name state](#).

↪ **U+0022 QUOTATION MARK (")**

↪ **U+0027 APOSTROPHE (')**

↪ **U+003C LESS-THAN SIGN (<)**

[parse error](#). Treat it as per the "anything else" entry below.

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Reconsume the EOF character.

↪ **Anything else**

Start a new attribute in the current tag token. Set that attribute's name to the [current input character](#), and its value to the empty string. Switch to the [§8.2.4.35 Attribute name state](#).

§ 8.2.4.37. Before attribute value state

Consume the [next input character](#):

↪ **U+0009 CHARACTER TABULATION (tab)**

↪ **U+000A LINE FEED (LF)**

↪ **U+000C FORM FEED (FF)**

↪ **U+0020 SPACE**

Ignore the character.

↪ **U+0022 QUOTATION MARK (")**

Switch to the [§8.2.4.38 Attribute value \(double-quoted\) state](#).

↪ **U+0026 AMPERSAND (&)**

Switch to the [§8.2.4.40 Attribute value \(unquoted\) state](#). Reconsume the [current input character](#).

↪ **U+0027 APOSTROPHE (')**

Switch to the [§8.2.4.39 Attribute value \(single-quoted\) state](#).

↪ **U+0000 NULL**

[parse error](#). Append a U+FFFD REPLACEMENT CHARACTER character to the current attribute's value. Switch to the [§8.2.4.40 Attribute value \(unquoted\) state](#).

↪ **U+003E GREATER-THAN SIGN (>)**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Emit the current tag token.

↪ **U+003C LESS-THAN SIGN (<)**

↪ **U+003D EQUALS SIGN (=)**

↪ **U+0060 GRAVE ACCENT (`)**

[parse error](#). Treat it as per the "anything else" entry below.

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Reconsume the EOF character.

↪ **Anything else**

Append the [current input character](#) to the current attribute's value. Switch to the [§8.2.4.40 Attribute value \(unquoted\) state](#).

§ 8.2.4.38. Attribute value (double-quoted) state

Consume the [next input character](#):

↪ **U+0022 QUOTATION MARK (")**

Switch to the [§8.2.4.42 After attribute value \(quoted\) state](#).

↪ **U+0026 AMPERSAND (&)**

Switch to the [§8.2.4.41 Character reference in attribute value state](#), with the [additional allowed character](#) being U+0022 QUOTATION MARK (").

↪ **U+0000 NULL**

[parse error](#). Append a U+FFFD REPLACEMENT CHARACTER character to the current attribute's value.

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Reconsume the EOF character.

↪ **Anything else**

Append the [current input character](#) to the current attribute's value.

§ 8.2.4.39. Attribute value (single-quoted) state

Consume the [next input character](#):

↪ **U+0027 APOSTROPHE (')**

Switch to the [§8.2.4.42 After attribute value \(quoted\) state](#).

↪ **U+0026 AMPERSAND (&)**

Switch to the [§8.2.4.41 Character reference in attribute value state](#), with the [additional allowed character](#) being U+0027 APOSTROPHE (').

↪ **U+0000 NULL**

[parse error](#). Append a U+FFFD REPLACEMENT CHARACTER character to the current attribute's value.

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Reconsume the EOF character.

↪ **Anything else**

Append the [current input character](#) to the current attribute's value.

§ 8.2.4.40. Attribute value (unquoted) state

Consume the [next input character](#):

- ↪ U+0009 CHARACTER TABULATION (tab)
- ↪ U+000A LINE FEED (LF)
- ↪ U+000C FORM FEED (FF)
- ↪ U+0020 SPACE

Switch to the [§8.2.4.34 Before attribute name state](#).

- ↪ U+0026 AMPERSAND (&)

Switch to the [§8.2.4.41 Character reference in attribute value state](#), with the [additional allowed character](#) being U+003E GREATER-THAN SIGN (>).

- ↪ U+003E GREATER-THAN SIGN (>)

Switch to the [§8.2.4.1 Data state](#). Emit the current tag token.

- ↪ U+0000 NULL

[parse error](#). Append a U+FFFD REPLACEMENT CHARACTER character to the current attribute's value.

- ↪ U+0022 QUOTATION MARK (")
- ↪ U+0027 APOSTROPHE (')
- ↪ U+003C LESS-THAN SIGN (<)
- ↪ U+003D EQUALS SIGN (=)
- ↪ U+0060 GRAVE ACCENT (`)

[parse error](#). Treat it as per the "anything else" entry below.

- ↪ EOF

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Reconsume the EOF character.

- ↪ Anything else

Append the [current input character](#) to the current attribute's value.

§ 8.2.4.41. Character reference in attribute value state

Attempt to [consume a character reference](#).

If nothing is returned, append a U+0026 AMPERSAND character (&) to the current attribute's value.

Otherwise, append the returned character tokens to the current attribute's value.

Finally, switch back to the attribute value state that switched into this state.

§ 8.2.4.42. After attribute value (quoted) state

Consume the [next input character](#):

- ↪ U+0009 CHARACTER TABULATION (tab)
- ↪ U+000A LINE FEED (LF)
- ↪ U+000C FORM FEED (FF)
- ↪ U+0020 SPACE

Switch to the [§8.2.4.34 Before attribute name state](#).

- ↪ U+002F SOLIDUS (/)

Switch to the [§8.2.4.43 Self-closing start tag state](#).

- ↪ U+003E GREATER-THAN SIGN (>)

Switch to the [§8.2.4.1 Data state](#). Emit the current tag token.

- ↪ EOF

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Reconsume the EOF character.

↪ Anything else

[parse error](#). Switch to the [§8.2.4.34 Before attribute name state](#). Reconsume the character.

§ 8.2.4.43. *Self-closing start tag state*

Consume the [next input character](#):

↪ U+003E GREATER-THAN SIGN (>)

Set the *self-closing flag* of the current tag token. Switch to the [§8.2.4.1 Data state](#). Emit the current tag token.

↪ EOF

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Reconsume the EOF character.

↪ Anything else

[parse error](#). Switch to the [§8.2.4.34 Before attribute name state](#). Reconsume the character.

§ 8.2.4.44. *Bogus comment state*

Consume every character up to and including the first U+003E GREATER-THAN SIGN character (>) or the end of the file (EOF), whichever comes first. If more than one character was consumed, then emit a comment token whose data is the concatenation of all the characters starting from and including the character that caused the state machine to switch into the bogus comment state, up to and including the character immediately before the last consumed character (i.e., up to the character just before the U+003E or EOF character), but with any U+0000 NULL characters replaced by U+FFFD REPLACEMENT CHARACTER characters. (If the comment was started by the end of the file (EOF), the token is empty. Similarly, the token is empty if it was generated by the string "<!>".)

Switch to the [§8.2.4.1 Data state](#).

If the end of the file was reached, reconsume the EOF character.

§ 8.2.4.45. *Markup declaration open state*

If the next two characters are both U+002D HYPHEN-MINUS characters (-), consume those two characters, create a comment token whose data is the empty string, and switch to the [§8.2.4.46 Comment start state](#).

Otherwise, if the next seven characters are an [ASCII case-insensitive](#) match for the word "DOCTYPE", then consume those characters and switch to the [§8.2.4.52 DOCTYPE state](#).

Otherwise, if there is an [adjusted current node](#) and it is not an element in the [HTML namespace](#) and the next seven characters are a [case-sensitive](#) match for the string "[CDATA[" (the five uppercase letters "CDATA" with a U+005B LEFT SQUARE BRACKET character before and after), then consume those characters and switch to the [§8.2.4.68 CDATA section state](#).

Otherwise, this is a [parse error](#). Switch to the [§8.2.4.44 Bogus comment state](#). The next character that is consumed, if any, is the first character that will be in the comment.

§ 8.2.4.46. *Comment start state*

Consume the [next input character](#):

↪ U+002D HYPHEN-MINUS (-)

Switch to the [§8.2.4.47 Comment start dash state](#).

↪ U+0000 NULL

[parse error](#). Append a U+FFFD REPLACEMENT CHARACTER character to the comment token's data. Switch to the [§8.2.4.48 Comment state](#).

↪ U+003E GREATER-THAN SIGN (>)

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Emit the comment token.

↪ EOF

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Emit the comment token. Reconsume the EOF character.

↪ Anything else

Append the [current input character](#) to the comment token's data. Switch to the [§8.2.4.48 Comment state](#).

§ 8.2.4.47. *Comment start dash state*

Consume the [next input character](#):

↪ U+002D HYPHEN-MINUS (-)

Switch to the [§8.2.4.50 Comment end state](#)

↪ U+0000 NULL

[parse error](#). Append a U+002D HYPHEN-MINUS character (-) and a U+FFFD REPLACEMENT CHARACTER character to the comment token's data. Switch to the [§8.2.4.48 Comment state](#).

↪ U+003E GREATER-THAN SIGN (>)

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Emit the comment token.

↪ EOF

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Emit the comment token. Reconsume the EOF character.

↪ Anything else

Append a U+002D HYPHEN-MINUS character (-) and the [current input character](#) to the comment token's data. Switch to the [§8.2.4.48 Comment state](#).

§ 8.2.4.48. *Comment state*

Consume the [next input character](#):

↪ U+002D HYPHEN-MINUS (-)

Switch to the [§8.2.4.49 Comment end dash state](#)

↪ U+0000 NULL

[parse error](#). Append a U+FFFD REPLACEMENT CHARACTER character to the comment token's data.

↪ EOF

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Emit the comment token. Reconsume the EOF character.

↪ Anything else

Append the [current input character](#) to the comment token's data.

§ 8.2.4.49. *Comment end dash state*

Consume the [next input character](#):

↪ U+002D HYPHEN-MINUS (-)

Switch to the [§8.2.4.50 Comment end state](#)

↪ U+0000 NULL

[parse error](#). Append a U+002D HYPHEN-MINUS character (-) and a U+FFFD REPLACEMENT CHARACTER character to the comment token's data. Switch to the [§8.2.4.48 Comment state](#).

↪ EOF

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Emit the comment token. Reconsume the EOF character.

↪ Anything else

Append a U+002D HYPHEN-MINUS character (-) and the [current input character](#) to the comment token's data.

Switch to the [§8.2.4.48 Comment state](#).

§ 8.2.4.50. *Comment end state*

Consume the [next input character](#):

↪ **U+003E GREATER-THAN SIGN (>)**

Switch to the [§8.2.4.1 Data state](#). Emit the comment token.

↪ **U+0000 NULL**

[parse error](#). Append two U+002D HYPHEN-MINUS characters (-) and a U+FFFD REPLACEMENT CHARACTER character to the comment token's data. Switch to the [§8.2.4.48 Comment state](#).

↪ **U+0021 EXCLAMATION MARK (!)**

[parse error](#). Switch to the [§8.2.4.51 Comment end bang state](#).

↪ **U+002D HYPHEN-MINUS (-)**

[parse error](#). Append a U+002D HYPHEN-MINUS character (-) to the comment token's data.

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Emit the comment token. Reconsume the EOF character.

↪ **Anything else**

[parse error](#). Append two U+002D HYPHEN-MINUS characters (-) and the [current input character](#) to the comment token's data. Switch to the [§8.2.4.48 Comment state](#).

§ 8.2.4.51. *Comment end bang state*

Consume the [next input character](#):

↪ **U+002D HYPHEN-MINUS (-)**

Append two U+002D HYPHEN-MINUS characters (-) and a U+0021 EXCLAMATION MARK character (!) to the comment token's data. Switch to the [§8.2.4.49 Comment end dash state](#).

↪ **U+003E GREATER-THAN SIGN (>)**

Switch to the [§8.2.4.1 Data state](#). Emit the comment token.

↪ **U+0000 NULL**

[parse error](#). Append two U+002D HYPHEN-MINUS characters (-), a U+0021 EXCLAMATION MARK character (!), and a U+FFFD REPLACEMENT CHARACTER character to the comment token's data. Switch to the [§8.2.4.48 Comment state](#).

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Emit the comment token. Reconsume the EOF character.

↪ **Anything else**

Append two U+002D HYPHEN-MINUS characters (-), a U+0021 EXCLAMATION MARK character (!), and the [current input character](#) to the comment token's data. Switch to the [§8.2.4.48 Comment state](#).

§ 8.2.4.52. *DOCTYPE state*

Consume the [next input character](#):

↪ **U+0009 CHARACTER TABULATION (tab)**

↪ **U+000A LINE FEED (LF)**

↪ **U+000C FORM FEED (FF)**

↪ **U+0020 SPACE**

Switch to the [§8.2.4.53 Before DOCTYPE name state](#).

↪ EOF

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Create a new DOCTYPE token. Set its *force-quirks flag* to *on*. Emit the token. Reconsume the EOF character.

↪ Anything else

[parse error](#). Switch to the [§8.2.4.53 Before DOCTYPE name state](#). Reconsume the character.

§ 8.2.4.53. Before DOCTYPE name state

Consume the [next input character](#):

↪ U+0009 CHARACTER TABULATION (tab)

↪ U+000A LINE FEED (LF)

↪ U+000C FORM FEED (FF)

↪ U+0020 SPACE

Ignore the character.

↪ [Uppercase ASCII letter](#)

Create a new DOCTYPE token. Set the token's name to the lowercase version of the [current input character](#) (add 0x0020 to the character's code point). Switch to the [§8.2.4.54 DOCTYPE name state](#).

↪ U+0000 NULL

[parse error](#). Create a new DOCTYPE token. Set the token's name to a U+FFFD REPLACEMENT CHARACTER character. Switch to the [§8.2.4.54 DOCTYPE name state](#).

↪ U+003E GREATER-THAN SIGN (>)

[parse error](#). Create a new DOCTYPE token. Set its *force-quirks flag* to *on*. Switch to the [§8.2.4.1 Data state](#). Emit the token.

↪ EOF

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Create a new DOCTYPE token. Set its *force-quirks flag* to *on*. Emit the token. Reconsume the EOF character.

↪ Anything else

Create a new DOCTYPE token. Set the token's name to the [current input character](#). Switch to the [§8.2.4.54 DOCTYPE name state](#).

§ 8.2.4.54. DOCTYPE name state

Consume the [next input character](#):

↪ U+0009 CHARACTER TABULATION (tab)

↪ U+000A LINE FEED (LF)

↪ U+000C FORM FEED (FF)

↪ U+0020 SPACE

Switch to the [§8.2.4.55 After DOCTYPE name state](#).

↪ U+003E GREATER-THAN SIGN (>)

Switch to the [§8.2.4.1 Data state](#). Emit the current DOCTYPE token.

↪ [Uppercase ASCII letter](#)

Append the lowercase version of the [current input character](#) (add 0x0020 to the character's code point) to the current DOCTYPE token's name.

↪ U+0000 NULL

[parse error](#). Append a U+FFFD REPLACEMENT CHARACTER character to the current DOCTYPE token's name.

↪ EOF

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Emit that DOCTYPE token. Reconsume the EOF character.

↪ **Anything else**

Append the [current input character](#) to the current DOCTYPE token's name.

§ 8.2.4.55. *After DOCTYPE name state*

Consume the [next input character](#):

↪ **U+0009 CHARACTER TABULATION (tab)**

↪ **U+000A LINE FEED (LF)**

↪ **U+000C FORM FEED (FF)**

↪ **U+0020 SPACE**

Ignore the character.

↪ **U+003E GREATER-THAN SIGN (>)**

Switch to the [§8.2.4.1 Data state](#). Emit the current DOCTYPE token.

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Emit that DOCTYPE token. Reconsume the EOF character.

↪ **Anything else**

If the six characters starting from the [current input character](#) are an [ASCII case-insensitive](#) match for the word "PUBLIC", then consume those characters and switch to the [§8.2.4.56 After DOCTYPE public keyword state](#).

Otherwise, if the six characters starting from the [current input character](#) are an [ASCII case-insensitive](#) match for the word "SYSTEM", then consume those characters and switch to the [§8.2.4.62 After DOCTYPE system keyword state](#).

Otherwise, this is a [parse error](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Switch to the [§8.2.4.67 Bogus DOCTYPE state](#).

§ 8.2.4.56. *After DOCTYPE public keyword state*

Consume the [next input character](#):

↪ **U+0009 CHARACTER TABULATION (tab)**

↪ **U+000A LINE FEED (LF)**

↪ **U+000C FORM FEED (FF)**

↪ **U+0020 SPACE**

Switch to the [§8.2.4.57 Before DOCTYPE public identifier state](#).

↪ **U+0022 QUOTATION MARK (")**

[parse error](#). Set the DOCTYPE token's public identifier to the empty string (not missing), then switch to the [§8.2.4.58 DOCTYPE public identifier \(double-quoted\) state](#).

↪ **U+0027 APOSTROPHE (')**

[parse error](#). Set the DOCTYPE token's public identifier to the empty string (not missing), then switch to the [§8.2.4.59 DOCTYPE public identifier \(single-quoted\) state](#).

↪ **U+003E GREATER-THAN SIGN (>)**

[parse error](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Switch to the [§8.2.4.1 Data state](#). Emit that DOCTYPE token.

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Emit that DOCTYPE token. Reconsume the EOF character.

↪ **Anything else**

[parse error](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Switch to the [§8.2.4.67 Bogus DOCTYPE state](#).

§ 8.2.4.57. *Before DOCTYPE public identifier state*

Consume the [next input character](#):

↪ **U+0009 CHARACTER TABULATION (tab)**

↪ **U+000A LINE FEED (LF)**

↪ **U+000C FORM FEED (FF)**

↪ **U+0020 SPACE**

Ignore the character.

↪ **U+0022 QUOTATION MARK (")**

Set the DOCTYPE token's public identifier to the empty string (not missing), then switch to the [§8.2.4.58 DOCTYPE public identifier \(double-quoted\) state](#).

↪ **U+0027 APOSTROPHE (')**

Set the DOCTYPE token's public identifier to the empty string (not missing), then switch to the [§8.2.4.59 DOCTYPE public identifier \(single-quoted\) state](#).

↪ **U+003E GREATER-THAN SIGN (>)**

[parse error](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Switch to the [§8.2.4.1 Data state](#). Emit that DOCTYPE token.

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Emit that DOCTYPE token. Reconsume the EOF character.

↪ **Anything else**

[parse error](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Switch to the [§8.2.4.67 Bogus DOCTYPE state](#).

§ 8.2.4.58. *DOCTYPE public identifier (double-quoted) state*

Consume the [next input character](#):

↪ **U+0022 QUOTATION MARK (")**

Switch to the [§8.2.4.60 After DOCTYPE public identifier state](#).

↪ **U+0000 NULL**

[parse error](#). Append a U+FFFD REPLACEMENT CHARACTER character to the current DOCTYPE token's public identifier.

↪ **U+003E GREATER-THAN SIGN (>)**

[parse error](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Switch to the [§8.2.4.1 Data state](#). Emit that DOCTYPE token.

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Emit that DOCTYPE token. Reconsume the EOF character.

↪ **Anything else**

Append the [current input character](#) to the current DOCTYPE token's public identifier.

§ 8.2.4.59. *DOCTYPE public identifier (single-quoted) state*

Consume the [next input character](#):

↪ **U+0027 APOSTROPHE (')**

Switch to the [§8.2.4.60 After DOCTYPE public identifier state](#).

↪ **U+0000 NULL**

[parse error](#). Append a U+FFFD REPLACEMENT CHARACTER character to the current DOCTYPE token's public identifier.

↪ **U+003E GREATER-THAN SIGN (>)**

[parse error](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Switch to the [§8.2.4.1 Data state](#). Emit that DOCTYPE token.

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Emit that DOCTYPE token. Reconsume the EOF character.

↪ **Anything else**

Append the [current input character](#) to the current DOCTYPE token's public identifier.

§ 8.2.4.60. *After DOCTYPE public identifier state*

Consume the [next input character](#):

↪ **U+0009 CHARACTER TABULATION (tab)**

↪ **U+000A LINE FEED (LF)**

↪ **U+000C FORM FEED (FF)**

↪ **U+0020 SPACE**

Switch to the [§8.2.4.61 Between DOCTYPE public and system identifiers state](#).

↪ **U+003E GREATER-THAN SIGN (>)**

Switch to the [§8.2.4.1 Data state](#). Emit the current DOCTYPE token.

↪ **U+0022 QUOTATION MARK (")**

[parse error](#). Set the DOCTYPE token's system identifier to the empty string (not missing), then switch to the [§8.2.4.64 DOCTYPE system identifier \(double-quoted\) state](#).

↪ **U+0027 APOSTROPHE (')**

[parse error](#). Set the DOCTYPE token's system identifier to the empty string (not missing), then switch to the [§8.2.4.65 DOCTYPE system identifier \(single-quoted\) state](#).

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Emit that DOCTYPE token. Reconsume the EOF character.

↪ **Anything else**

[parse error](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Switch to the [§8.2.4.67 Bogus DOCTYPE state](#).

§ 8.2.4.61. *Between DOCTYPE public and system identifiers state*

Consume the [next input character](#):

↪ **U+0009 CHARACTER TABULATION (tab)**

↪ **U+000A LINE FEED (LF)**

↪ **U+000C FORM FEED (FF)**

↪ **U+0020 SPACE**

Ignore the character.

↪ **U+003E GREATER-THAN SIGN (>)**

Switch to the [§8.2.4.1 Data state](#). Emit the current DOCTYPE token.

↪ **U+0022 QUOTATION MARK (")**

Set the DOCTYPE token's system identifier to the empty string (not missing), then switch to the [§8.2.4.64 DOCTYPE system identifier \(double-quoted\) state](#).

↪ **U+0027 APOSTROPHE (')**

Set the DOCTYPE token's system identifier to the empty string (not missing), then switch to the [§8.2.4.65 DOCTYPE system identifier \(single-quoted\) state](#).

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Emit that DOCTYPE token. Reconsume the EOF character.

↪ **Anything else**

[parse error](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Switch to the [§8.2.4.67 Bogus DOCTYPE state](#).

§ 8.2.4.62. *After DOCTYPE system keyword state*

Consume the [next input character](#):

↪ **U+0009 CHARACTER TABULATION (tab)**

↪ **U+000A LINE FEED (LF)**

↪ **U+000C FORM FEED (FF)**

↪ **U+0020 SPACE**

Switch to the [§8.2.4.63 Before DOCTYPE system identifier state](#).

↪ **U+0022 QUOTATION MARK (")**

[parse error](#). Set the DOCTYPE token's system identifier to the empty string (not missing), then switch to the [§8.2.4.64 DOCTYPE system identifier \(double-quoted\) state](#).

↪ **U+0027 APOSTROPHE (')**

[parse error](#). Set the DOCTYPE token's system identifier to the empty string (not missing), then switch to the [§8.2.4.65 DOCTYPE system identifier \(single-quoted\) state](#).

↪ **U+003E GREATER-THAN SIGN (>)**

[parse error](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Switch to the [§8.2.4.1 Data state](#). Emit that DOCTYPE token.

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Emit that DOCTYPE token. Reconsume the EOF character.

↪ **Anything else**

[parse error](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Switch to the [§8.2.4.67 Bogus DOCTYPE state](#).

§ 8.2.4.63. *Before DOCTYPE system identifier state*

Consume the [next input character](#):

↪ **U+0009 CHARACTER TABULATION (tab)**

↪ **U+000A LINE FEED (LF)**

↪ **U+000C FORM FEED (FF)**

↪ **U+0020 SPACE**

Ignore the character.

↪ **U+0022 QUOTATION MARK (")**

Set the DOCTYPE token's system identifier to the empty string (not missing), then switch to the [§8.2.4.64 DOCTYPE system identifier \(double-quoted\) state](#).

↪ **U+0027 APOSTROPHE (')**

Set the DOCTYPE token's system identifier to the empty string (not missing), then switch to the [§8.2.4.65 DOCTYPE system identifier \(single-quoted\) state](#).

↪ **U+003E GREATER-THAN SIGN (>)**

[parse error](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Switch to the [§8.2.4.1 Data state](#). Emit that DOCTYPE token.

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Emit that DOCTYPE token. Reconsume the EOF character.

↪ **Anything else**

[parse error](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Switch to the [§8.2.4.67 Bogus DOCTYPE state](#).

§ 8.2.4.64. *DOCTYPE system identifier (double-quoted) state*

Consume the [next input character](#):

↪ **U+0022 QUOTATION MARK (")**

Switch to the [§8.2.4.66 After DOCTYPE system identifier state](#).

↪ **U+0000 NULL**

[parse error](#). Append a U+FFFD REPLACEMENT CHARACTER character to the current DOCTYPE token's system identifier.

↪ **U+003E GREATER-THAN SIGN (>)**

[parse error](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Switch to the [§8.2.4.1 Data state](#). Emit that DOCTYPE token.

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Emit that DOCTYPE token. Reconsume the EOF character.

↪ **Anything else**

Append the [current input character](#) to the current DOCTYPE token's system identifier.

§ 8.2.4.65. *DOCTYPE system identifier (single-quoted) state*

Consume the [next input character](#):

↪ **U+0027 APOSTROPHE (')**

Switch to the [§8.2.4.66 After DOCTYPE system identifier state](#).

↪ **U+0000 NULL**

[parse error](#). Append a U+FFFD REPLACEMENT CHARACTER character to the current DOCTYPE token's system identifier.

↪ **U+003E GREATER-THAN SIGN (>)**

[parse error](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Switch to the [§8.2.4.1 Data state](#). Emit that DOCTYPE token.

↪ **EOF**

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Emit that DOCTYPE token. Reconsume the EOF character.

↪ **Anything else**

Append the [current input character](#) to the current DOCTYPE token's system identifier.

§ 8.2.4.66. *After DOCTYPE system identifier state*

Consume the [next input character](#):

- ↪ U+0009 CHARACTER TABULATION (tab)
- ↪ U+000A LINE FEED (LF)
- ↪ U+000C FORM FEED (FF)
- ↪ U+0020 SPACE

Ignore the character.

- ↪ U+003E GREATER-THAN SIGN (>)

Switch to the [§8.2.4.1 Data state](#). Emit the current DOCTYPE token.

- ↪ EOF

[parse error](#). Switch to the [§8.2.4.1 Data state](#). Set the DOCTYPE token's *force-quirks flag* to *on*. Emit that DOCTYPE token. Reconsume the EOF character.

- ↪ Anything else

[parse error](#). Switch to the [§8.2.4.67 Bogus DOCTYPE state](#). (This does *not* set the DOCTYPE token's *force-quirks flag* to *on*.)

§ 8.2.4.67. Bogus DOCTYPE state

Consume the [next input character](#):

- ↪ U+003E GREATER-THAN SIGN (>)

Switch to the [§8.2.4.1 Data state](#). Emit the DOCTYPE token.

- ↪ EOF

Switch to the [§8.2.4.1 Data state](#). Emit the DOCTYPE token. Reconsume the EOF character.

- ↪ Anything else

Ignore the character.

§ 8.2.4.68. CDATA section state

Switch to the [§8.2.4.1 Data state](#).

Consume every character up to the next occurrence of the three character sequence U+005D RIGHT SQUARE BRACKET U+005D RIGHT SQUARE BRACKET U+003E GREATER-THAN SIGN (]]>), or the end of the file (EOF), whichever comes first. Emit a series of character tokens consisting of all the characters consumed except the matching three character sequence at the end (if one was found before the end of the file).

If the end of the file was reached, reconsume the EOF character.

§ 8.2.4.69. Tokenizing character references

This section defines how to **consume a character reference**, optionally with an **additional allowed character**, which, if specified where the algorithm is invoked, adds a character to the list of characters that cause there to not be a character reference.

This definition is used when parsing character references in text and in attributes.

The behavior depends on the identity of the next character (the one immediately after the U+0026 AMPERSAND character), as follows:

- ↪ U+0009 CHARACTER TABULATION (tab)
- ↪ U+000A LINE FEED (LF)
- ↪ U+000C FORM FEED (FF)
- ↪ U+0020 SPACE
- ↪ U+003C LESS-THAN SIGN

↪ **U+0026 AMPERSAND**

↪ **EOF**

↪ **The additional allowed character, if there is one**

Not a character reference. No characters are consumed, and nothing is returned. (This is not an error, either.)

↪ **U+0023 NUMBER SIGN (#)**

Consume the U+0023 NUMBER SIGN.

The behavior further depends on the character after the U+0023 NUMBER SIGN:

↪ **U+0078 LATIN SMALL LETTER X**

↪ **U+0058 LATIN CAPITAL LETTER X**

Consume the X.

Follow the steps below, but using [ASCII hex digits](#).

When it comes to interpreting the number, interpret it as a hexadecimal number.

↪ **Anything else**

Follow the steps below, but using [ASCII digits](#).

When it comes to interpreting the number, interpret it as a decimal number.

Consume as many characters as match the range of characters given above ([ASCII hex digits](#) or [ASCII digits](#)).

If no characters match the range, then don't consume any characters (and unconsume the U+0023 NUMBER SIGN character and, if appropriate, the X character). This is a [parse error](#); nothing is returned.

Otherwise, if the next character is a U+003B SEMICOLON, consume that too. If it isn't, there is a [parse error](#).

If one or more characters match the range, then take them all and interpret the string of characters as a number (either hexadecimal or decimal as appropriate).

If that number is one of the numbers in the first column of the following table, then this is a [parse error](#). Find the row with that number in the first column, and return a character token for the Unicode character given in the second column of that row.

Number	Unicode character	
0x00	U+FFFD	REPLACEMENT CHARACTER
0x80	U+20AC	EURO SIGN (€)
0x82	U+201A	SINGLE LOW-9 QUOTATION MARK (,))
0x83	U+0192	LATIN SMALL LETTER F WITH HOOK (<i>f</i>)
0x84	U+201E	DOUBLE LOW-9 QUOTATION MARK (,,)
0x85	U+2026	HORIZONTAL ELLIPSIS (...)
0x86	U+2020	DAGGER (†)
0x87	U+2021	DOUBLE DAGGER (‡)
0x88	U+02C6	MODIFIER LETTER CIRCUMFLEX ACCENT (ˆ)
0x89	U+2030	PER MILLE SIGN (‰)
0x8A	U+0160	LATIN CAPITAL LETTER S WITH CARON (Š)
0x8B	U+2039	SINGLE LEFT-POINTING ANGLE QUOTATION MARK (‹)
0x8C	U+0152	LATIN CAPITAL LIGATURE OE (Œ)
0x8E	U+017D	LATIN CAPITAL LETTER Z WITH CARON (Ž)
0x91	U+2018	LEFT SINGLE QUOTATION MARK (‘)
0x92	U+2019	RIGHT SINGLE QUOTATION MARK (’)
0x93	U+201C	LEFT DOUBLE QUOTATION MARK (“)
0x94	U+201D	RIGHT DOUBLE QUOTATION MARK (”)

Number	Unicode character	
0x95	U+2022	BULLET (•)
0x96	U+2013	EN DASH (–)
0x97	U+2014	EM DASH (—)
0x98	U+02DC	SMALL TILDE (˘)
0x99	U+2122	TRADE MARK SIGN (™)
0x9A	U+0161	LATIN SMALL LETTER S WITH CARON (š)
0x9B	U+203A	SINGLE RIGHT-POINTING ANGLE QUOTATION MARK (›)
0x9C	U+0153	LATIN SMALL LIGATURE OE (œ)
0x9E	U+017E	LATIN SMALL LETTER Z WITH CARON (ž)
0x9F	U+0178	LATIN CAPITAL LETTER Y WITH DIAERESIS (ÿ)

Otherwise, if the number is in the range 0xD800 to 0xDFFF or is greater than 0x10FFFF, then this is a [parse error](#). Return a U+FFFD REPLACEMENT CHARACTER character token.

Otherwise, return a character token for the Unicode character whose code point is that number.

Additionally, if the number is in the range 0x0001 to 0x0008, 0x000D to 0x001F, 0x007F to 0x009F, 0xFDD0 to 0xFDEF, or is one of 0x000B, 0xFFFE, 0xFFFF, 0x1FFE, 0x1FFF, 0x2FFE, 0x2FFF, 0x3FFE, 0x3FFF, 0x4FFE, 0x4FFF, 0x5FFE, 0x5FFF, 0x6FFE, 0x6FFF, 0x7FFE, 0x7FFF, 0x8FFE, 0x8FFF, 0x9FFE, 0x9FFF, 0xAFFE, 0xAFFF, 0xBFFE, 0xBFFF, 0xCFFE, 0xCFFF, 0xDFFE, 0xDFFF, 0xEFFE, 0xEFFF, 0xFFFE, 0xFFFF, 0x10FFE, or 0x10FFF, then this is a [parse error](#).

↪ Anything else

Consume the maximum number of characters possible, with the consumed characters matching one of the identifiers in the first column of the [§8.5 Named character references](#) table (in a [case-sensitive](#) manner).

If no match can be made, then no characters are consumed, and nothing is returned. In this case, if the characters after the U+0026 AMPERSAND character (&) consist of a sequence of one or more [alphanumeric ASCII characters](#) followed by a U+003B SEMICOLON character (;), then this is a [parse error](#).

If the character reference is being consumed [as part of an attribute](#), and the last character matched is not a U+003B SEMICOLON character (;), and the next character is either a U+003D EQUALS SIGN character (=) or an [alphanumeric ASCII character](#), then, for historical reasons, all the characters that were matched after the U+0026 AMPERSAND character (&) must be unconsumed, and nothing is returned.

However, if this next character is in fact a U+003D EQUALS SIGN character (=), then this is a [parse error](#), because some legacy user agents will misinterpret the markup in those cases.

Otherwise, a character reference is parsed. If the last character matched is not a U+003B SEMICOLON character (;), there is a [parse error](#).

Return one or two character tokens for the character(s) corresponding to the character reference name (as given by the second column of the [§8.5 Named character references](#) table).

EXAMPLE 647

If the markup contains (not in an attribute) the string I'm &notit; I tell you, the character reference is parsed as "not", as in, I'm -it; I tell you (and this is a parse error). But if the markup was I'm ∉ I tell you, the character reference would be parsed as "notin;", resulting in I'm ∉ I tell you (and no parse error).

§ 8.2.5. Tree construction

The input to the tree construction stage is a sequence of tokens from the [tokenization](#) stage. The tree construction stage is associated with a DOM Document object when a parser is created. The "output" of this stage consists of dynamically modifying or extending that document's DOM tree.

This specification does not define when an interactive user agent has to render the Document so that it is available to the user, or when it has to begin accepting user input.



As each token is emitted from the tokenizer, the user agent must follow the appropriate steps from the following list, known as the **tree construction dispatcher**:

- ↪ If the [stack of open elements](#) is empty
- ↪ If the [adjusted current node](#) is an element in the [HTML namespace](#)
- ↪ If the [adjusted current node](#) is a [MathML text integration point](#) and the token is a start tag whose tag name is neither "mglyph" nor "malignmark"
- ↪ If the [adjusted current node](#) is a [MathML text integration point](#) and the token is a character token
- ↪ If the [adjusted current node](#) is an `<annotation-xml>` element in the [MathML namespace](#) and the token is a start tag whose tag name is "svg"
- ↪ If the [adjusted current node](#) is an [HTML integration point](#) and the token is a start tag
- ↪ If the [adjusted current node](#) is an [HTML integration point](#) and the token is a character token
- ↪ If the token is an end-of-file token
 - Process the token according to the rules given in the section corresponding to the current [insertion mode](#) in HTML content.
- ↪ Otherwise
 - Process the token according to the rules given in the section for parsing tokens [in foreign content](#).

The **next token** is the token that is about to be processed by the [tree construction dispatcher](#) (even if the token is subsequently just ignored).

A node is a **MathML text integration point** if it is one of the following elements:

- An `<mi>` element in the [MathML namespace](#)
- An `<mo>` element in the [MathML namespace](#)
- An `<mn>` element in the [MathML namespace](#)
- An `<ms>` element in the [MathML namespace](#)
- An `<mtext>` element in the [MathML namespace](#)

A node is an **HTML integration point** if it is one of the following elements:

- An `<annotation-xml>` element in the [MathML namespace](#) whose start tag token had an attribute with the name "encoding" whose value was an [ASCII case-insensitive](#) match for the string "text/html"
- An `<annotation-xml>` element in the [MathML namespace](#) whose start tag token had an attribute with the name "encoding" whose value was an [ASCII case-insensitive](#) match for the string "application/xhtml+xml"
- A `<foreignObject>` element in the [SVG namespace](#)
- A `<desc>` element in the [SVG namespace](#)
- A `<title>` element in the [SVG namespace](#)

NOTE:

If the node in question is the [context](#) element passed to the [HTML fragment parsing algorithm](#), then the start tag token for that element is the "fake" token created during by that [HTML fragment parsing algorithm](#).



NOTE:

Not all of the tag names mentioned below are conformant tag names in this specification; many are included to handle legacy content. They still form part of the algorithm that implementations are required to implement to claim conformance.

NOTE:

The algorithm described below places no limit on the depth of the DOM tree generated, or on the length of tag names, attribute names, attribute values, Text nodes, etc. While implementors are encouraged to avoid arbitrary limits, it is recognized that [practical concerns](#) will likely force user agents to impose nesting depth constraints.

§ 8.2.5.1. Creating and inserting nodes

While the parser is processing a token, it can enable or disable **foster parenting**. This affects the following algorithm.

The **appropriate place for inserting a node**, optionally using a particular *override target*, is the position in an element returned by running the following steps:

1. If there was an *override target* specified, then let *target* be the *override target*.

Otherwise, let *target* be the [current node](#).

2. Determine the *adjusted insertion location* using the first matching steps from the following list:

↪ If **foster parenting** is enabled and *target* is a [<table>](#), [<tbody>](#), [<tfoot>](#), [<thead>](#), or [<tr>](#) element

NOTE:

Foster parenting happens when content is misnested in tables.

Run these substeps:

1. Let *last template* be the last [<template>](#) element in the [stack of open elements](#), if any.
2. Let *last table* be the last [<table>](#) element in the [stack of open elements](#), if any.
3. If there is a *last template* and either there is no *last table*, or there is one, but *last template* is lower (more recently added) than *last table* in the [stack of open elements](#), then: let *adjusted insertion location* be inside *last template*'s [template contents](#), after its last child (if any), and abort these substeps.
4. If there is no *last table*, then let *adjusted insertion location* be inside the first element in the [stack of open elements](#) (the [<html>](#) element), after its last child (if any), and abort these substeps. ([fragment case](#))
5. If *last table* has a parent node, then let *adjusted insertion location* be inside *last table*'s parent node, immediately before *last table*, and abort these substeps.
6. Let *previous element* be the element immediately above *last table* in the [stack of open elements](#).
7. Let *adjusted insertion location* be inside *previous element*, after its last child (if any).

NOTE:

These steps are involved in part because it's possible for elements, the [<table>](#) element in this case in particular, to have been moved by a script around in the DOM, or indeed removed from the DOM entirely, after the element was inserted by the parser.

↪ Otherwise

Let *adjusted insertion location* be inside *target*, after its last child (if any).

3. If the *adjusted insertion location* is inside a [template](#) element, let it instead be inside the [<template>](#) element's [template contents](#), after its last child (if any).
4. Return the *adjusted insertion location*.



When the steps below require the user agent to **create an element for a token** in a particular *given namespace* and with a particular *intended parent*, the user agent must run the following steps:

1. Create a node implementing the interface appropriate for the element type corresponding to the tag name of the token in *given namespace* (as given in the specification that defines that element, e.g., for an [<a>](#) element in the [HTML namespace](#), this specification defines it to be the `HTMLAnchorElement` interface), with the tag name being the name of that element, with the node being in the given namespace, and with the attributes on the node being those given in the given token.

The interface appropriate for an element in the [HTML namespace](#) that is not defined in this specification (or [other applicable specifications](#)) is [HTMLUnknownElement](#). Elements in other namespaces whose interface is not defined by

that namespace’s specification must use the interface [Element](#).

The [node document](#) of the newly created element must be the [node document](#) of the *intended parent*.

2. If the newly created element has an `xmlns` attribute in the *XMLNS namespace* whose value is not exactly the same as the element’s namespace, that is a [parse error](#). Similarly, if the newly created element has an `xmlns:xlink` attribute in the *XMLNS namespace* whose value is not the *XLink namespace*, that is a [parse error](#).
3. If the newly created element is a [resettable element](#), invoke its [reset algorithm](#). (This initializes the element’s [value](#) and [checkedness](#) based on the element’s attributes.)
4. If the element is a [form-associated element](#), and the [form element pointer](#) is not null, and there is no template element on the [stack of open elements](#), and the newly created element is either not [reassociateable](#) or doesn’t have a `form` attribute, and the *intended parent* is in the same [home subtree](#) as the element pointed to by the [form element pointer](#), associate the newly created element with the `<form>` element pointed to by the [form element pointer](#), and suppress the running of the [reset the form owner](#) algorithm when the parser subsequently attempts to insert the element.
5. Return the newly created element.



When the steps below require the user agent to **insert a foreign element** for a token in a given namespace, the user agent must run these steps:

1. Let the *adjusted insertion location* be the [appropriate place for inserting a node](#).
2. [Create an element for the token](#) in the given namespace, with the intended parent being the element in which the *adjusted insertion location* finds itself.
3. If it is possible to insert an element at the *adjusted insertion location*, then insert the newly created element at the *adjusted insertion location*.

NOTE:

If the *adjusted insertion location* cannot accept more elements, e.g., because it’s a Document that already has an element child, then the newly created element is dropped on the floor.

4. Push the element onto the [stack of open elements](#) so that it is the new [current node](#).
5. Return the newly created element.

When the steps below require the user agent to **insert an HTML element** for a token, the user agent must [insert a foreign element](#) for the token, in the [HTML namespace](#).



When the steps below require the user agent to **adjust MathML attributes** for a token, then, if the token has an attribute named `definitionurl`, change its name to `definitionURL` (note the case difference).

When the steps below require the user agent to **adjust SVG attributes** for a token, then, for each attribute on the token whose attribute name is one of the ones in the first column of the following table, change the attribute’s name to the name given in the corresponding cell in the second column. (This fixes the case of SVG attributes that are not all lowercase.)

Attribute name on token	Attribute name on element
<code>attributename</code>	<code>attributeName</code>
<code>attributetype</code>	<code>attributeType</code>
<code>basefrequency</code>	<code>baseFrequency</code>
<code>baseprofile</code>	<code>baseProfile</code>
<code>calcmode</code>	<code>calcMode</code>
<code>clippathunits</code>	<code>clipPathUnits</code>

Attribute name on token	Attribute name on element
diffuseconstant	diffuseConstant
edgemode	edgeMode
filterunits	filterUnits
glyphref	glyphRef
gradienttransform	gradientTransform
gradientunits	gradientUnits
kernelmatrix	kernelMatrix
kernelunitlength	kernelUnitLength
keypoints	keyPoints
keysplines	keySplines
keytimes	keyTimes
lengthadjust	lengthAdjust
limitingconeangle	limitingConeAngle
markerheight	markerHeight
markerunits	markerUnits
markerwidth	markerWidth
maskcontentunits	maskContentUnits
maskunits	maskUnits
numoctaves	numOctaves
pathlength	pathLength
patterncontentunits	patternContentUnits
patterntransform	patternTransform
patternunits	patternUnits
pointsatx	pointsAtX
pointsaty	pointsAtY
pointsatz	pointsAtZ
preservealpha	preserveAlpha
preserveaspectratio	preserveAspectRatio
primitiveunits	primitiveUnits
refx	refX
refy	refY
repeatcount	repeatCount
repeatdur	repeatDur
requiredextensions	requiredExtensions
requiredfeatures	requiredFeatures
specularconstant	specularConstant
specularexponent	specularExponent
spreadmethod	spreadMethod
startoffset	startOffset
stddeviation	stdDeviation
stitchtiles	stitchTiles
surfacescale	surfaceScale
systemlanguage	systemLanguage
tablevalues	tableValues

Attribute name on token	Attribute name on element
targetx	targetX
targety	targetY
textlength	textLength
viewbox	viewBox
viewtarget	viewTarget
xchannelselector	xChannelSelector
ychannelselector	yChannelSelector
zoomandpan	zoomAndPan

When the steps below require the user agent to **adjust foreign attributes** for a token, then, if any of the attributes on the token match the strings given in the first column of the following table, let the attribute be a namespaced attribute, with the prefix being the string given in the corresponding cell in the second column, the local name being the string given in the corresponding cell in the third column, and the namespace being the namespace given in the corresponding cell in the fourth column. (This fixes the use of namespaced attributes, in particular [lang](#) attributes in the [XML namespace](#).)

Attribute name	Prefix	Local name	Namespace
xlink:actuate	xlink	actuate	XLink namespace
xlink:arcrole	xlink	arcrole	XLink namespace
xlink:href	xlink	href	XLink namespace
xlink:role	xlink	role	XLink namespace
xlink:show	xlink	show	XLink namespace
xlink:title	xlink	title	XLink namespace
xlink:type	xlink	type	XLink namespace
xml:lang	xml	lang	XML namespace
xml:space	xml	space	XML namespace
xmlns	(none)	xmlns	XMLNS namespace
xmlns:xlink	xmlns	xlink	XMLNS namespace



When the steps below require the user agent to **insert a character** while processing a token, the user agent must run the following steps:

1. Let *data* be the characters passed to the algorithm, or, if no characters were explicitly specified, the character of the character token being processed.
2. Let the *adjusted insertion location* be the [appropriate place for inserting a node](#).
3. If the *adjusted insertion location* is in a Document node, then abort these steps.

NOTE:

The DOM will not let Document nodes have Text node children, so they are dropped on the floor.

4. If there is a Text node immediately before the *adjusted insertion location*, then append *data* to that Text node's data.

Otherwise, create a new Text node whose data is *data* and whose [node document](#) is the same as that of the element in which the *adjusted insertion location* finds itself, and insert the newly created node at the *adjusted insertion location*.

EXAMPLE 648

Here are some sample inputs to the parser and the corresponding number of Text nodes that they result in, assuming a user agent that executes scripts.

Input	Number of Text nodes
<pre>A<script> var script = document.getElementsByTagName('script')[0]; document.body.removeChild(script); </script>B</pre>	One Text node in the document, containing "AB".
<pre>A<script> var text = document.createTextNode('B'); document.body.appendChild(text); </script>C</pre>	Three Text nodes; "A" before the script, the script's contents, and "BC" after the script (the parser appends to the Text node created by the script).
<pre>A<script> var text = document.getElementsByTagName('script') [0].firstChild; text.data = 'B'; document.body.appendChild(text); </script>C</pre>	Two adjacent Text nodes in the document, containing "A" and "BC".
<pre>A<table>B<tr>C</tr>D</table></pre>	One Text node before the table, containing "ABCD". (This is caused by foster parenting .)
<pre>A<table><tr> B</tr> C</table></pre>	One Text node before the table, containing "A B C" (A-space-B-space-C). (This is caused by foster parenting .)
<pre>A<table><tr> B</tr> C</table></pre>	One Text node before the table, containing "A BC" (A-space-B-C), and one Text node inside the table (as a child of a tbody) with a single space character. (Space characters separated from non-space characters by non-character tokens are not affected by foster parenting , even if those other tokens then get ignored.)



When the steps below require the user agent to **insert a comment** while processing a comment token, optionally with an explicitly insertion position *position*, the user agent must run the following steps:

1. Let *data* be the data given in the comment token being processed.
2. If *position* was specified, then let the *adjusted insertion location* be *position*. Otherwise, let *adjusted insertion location* be the [appropriate place for inserting a node](#).
3. Create a Comment node whose data attribute is set to *data* and whose [node document](#) is the same as that of the node in which the *adjusted insertion location* finds itself.
4. Insert the newly created node at the *adjusted insertion location*.



DOM mutation events must not fire for changes caused by the user agent parsing the document. This includes the parsing of any content inserted using `document.write()` and `document.writeln()` calls. [\[UIEVENTS\]](#)

However, [mutation observers](#) *do* fire, as required by the DOM specification.

§ 8.2.5.2. Parsing elements that contain only text

The **generic raw text element parsing algorithm** and the **generic RCDATA element parsing algorithm** consist of the following steps. These algorithms are always invoked in response to a start tag token.

1. [Insert an HTML element](#) for the token.
2. If the algorithm that was invoked is the [generic raw text element parsing algorithm](#), switch the tokenizer to the [§8.2.4.5 RAWTEXT state](#); otherwise the algorithm invoked was the [generic RCDATA element parsing algorithm](#), switch the tokenizer to the [§8.2.4.3 RCDATA state](#).
3. Let the [original insertion mode](#) be the current [insertion mode](#).
4. Then, switch the [insertion mode](#) to "text".

§ 8.2.5.3. Closing elements that have implied end tags

When the steps below require the user agent to **generate implied end tags**, then, while the [current node](#) is a `<dd>` element, a `<dt>` element, an `<li>` element, an `<option>` element, an `<optgroup>` element, a `<p>` element, an `<rb>` element, an `<rp>` element, an `rt` element, or an `<rtc>` element, the user agent must pop the [current node](#) off the [stack of open elements](#).

If a step requires the user agent to generate implied end tags but lists an element to exclude from the process, then the user agent must perform the above steps as if that element was not in the above list.

When the steps below require the user agent to **generate all implied end tags thoroughly**, then, while the [current node](#) is a `<caption>` element, a `<colgroup>` element, a `<dd>` element, a `<dt>` element, an `<li>` element, an `<optgroup>` element, an `<option>` element, a `<p>` element, an `<rb>` element, an `<rp>` element, an `rt` element, an `<rtc>` element, a `<tbody>` element, a `<td>` element, a `<tfoot>` element, a `<th>` element, a `<thead>` element, or a `<tr>` element, the user agent must pop the [current node](#) off the [stack of open elements](#).

§ 8.2.5.4. The rules for parsing tokens in HTML content

§ 8.2.5.4.1. THE "INITIAL" INSERTION MODE

When the user agent is to apply the rules for the ["initial" insertion mode](#), the user agent must handle the token as follows:

- ↪ **A character token that is one of U+0009 CHARACTER TABULATION, U+000A LINE FEED (LF), U+000C FORM FEED (FF), U+000D CARRIAGE RETURN (CR), or U+0020 SPACE**

Ignore the token.

- ↪ **A comment token**

[Insert a comment](#) as the last child of the Document object.

- ↪ **A DOCTYPE token**

If the DOCTYPE token's name is not a [case-sensitive](#) match for the string "html", or the token's public identifier is not missing, or the token's system identifier is neither missing nor a [case-sensitive](#) match for the string ["about:legacy-compat"](#), and none of the sets of conditions in the following list are matched, then there is a [parse error](#).

- The DOCTYPE token's name is a case-sensitive match for the string "html", the token's public identifier is the case-sensitive string "-//W3C//DTD HTML 4.0//EN", and the token's system identifier is either missing or the case-sensitive string "http://www.w3.org/TR/REC-html40/strict.dtd".

- The DOCTYPE token's name is a case-sensitive match for the string "html", the token's public identifier is the case-sensitive string "-//W3C//DTD HTML 4.01//EN", and the token's system identifier is either missing or the case-sensitive string "http://www.w3.org/TR/html4/strict.dtd".
- The DOCTYPE token's name is a case-sensitive match for the string "html", the token's public identifier is the case-sensitive string "-//W3C//DTD XHTML 1.0 Strict//EN", and the token's system identifier is the case-sensitive string "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd".
- The DOCTYPE token's name is a case-sensitive match for the string "html", the token's public identifier is the case-sensitive string "-//W3C//DTD XHTML 1.1//EN", and the token's system identifier is the case-sensitive string "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd".

Conformance checkers may, based on the values (including presence or lack thereof) of the DOCTYPE token's name, public identifier, or system identifier, switch to a conformance checking mode for another language (e.g., based on the DOCTYPE token a conformance checker could recognize that the document is an HTML 4.01-era document, and defer to an HTML 4.01 conformance checker.)

Append a `DocumentType` node to the `Document` node, with the `name` attribute set to the name given in the DOCTYPE token, or the empty string if the name was missing; the `publicId` attribute set to the public identifier given in the DOCTYPE token, or the empty string if the public identifier was missing; the `systemId` attribute set to the system identifier given in the DOCTYPE token, or the empty string if the system identifier was missing; and the other attributes specific to `DocumentType` objects set to null and empty lists as appropriate. Associate the `DocumentType` node with the `Document` object so that it is returned as the value of the `doctype` attribute of the `Document` object.

Then, if the document is *not* an [iframe srcdoc document](#), and the DOCTYPE token matches one of the conditions in the following list, then set the `Document` to [quirks mode](#):

- The *force-quirks flag* is set to *on*.
- The name is set to anything other than "html" (compared [case-sensitively](#)).
- The public identifier is set to: "-//W3O//DTD W3 HTML Strict 3.0//EN/"
- The public identifier is set to: "-//W3C//DTD HTML 4.0 Transitional/EN"
- The public identifier is set to: "HTML"
- The system identifier is set to: "http://www.ibm.com/data/dtd/v11/ibmxhtml1-transitional.dtd"
- The public identifier starts with: "+//Silmaril//dtd html Pro v0r11 19970101/"
- The public identifier starts with: "-//AS//DTD HTML 3.0 asWedit + extensions/"
- The public identifier starts with: "-//AdvaSoft Ltd//DTD HTML 3.0 asWedit + extensions/"
- The public identifier starts with: "-//IETF//DTD HTML 2.0 Level 1/"
- The public identifier starts with: "-//IETF//DTD HTML 2.0 Level 2/"
- The public identifier starts with: "-//IETF//DTD HTML 2.0 Strict Level 1/"
- The public identifier starts with: "-//IETF//DTD HTML 2.0 Strict Level 2/"
- The public identifier starts with: "-//IETF//DTD HTML 2.0 Strict/"
- The public identifier starts with: "-//IETF//DTD HTML 2.0/"
- The public identifier starts with: "-//IETF//DTD HTML 2.1E/"
- The public identifier starts with: "-//IETF//DTD HTML 3.0/"
- The public identifier starts with: "-//IETF//DTD HTML 3.2 Final/"
- The public identifier starts with: "-//IETF//DTD HTML 3.2/"
- The public identifier starts with: "-//IETF//DTD HTML 3/"
- The public identifier starts with: "-//IETF//DTD HTML Level 0/"
- The public identifier starts with: "-//IETF//DTD HTML Level 1/"
- The public identifier starts with: "-//IETF//DTD HTML Level 2/"
- The public identifier starts with: "-//IETF//DTD HTML Level 3/"
- The public identifier starts with: "-//IETF//DTD HTML Strict Level 0/"
- The public identifier starts with: "-//IETF//DTD HTML Strict Level 1/"
- The public identifier starts with: "-//IETF//DTD HTML Strict Level 2/"
- The public identifier starts with: "-//IETF//DTD HTML Strict Level 3/"
- The public identifier starts with: "-//IETF//DTD HTML Strict/"
- The public identifier starts with: "-//IETF//DTD HTML/"
- The public identifier starts with: "-//Metrius//DTD Metrius Presentational/"
- The public identifier starts with: "-//Microsoft//DTD Internet Explorer 2.0 HTML Strict/"
- The public identifier starts with: "-//Microsoft//DTD Internet Explorer 2.0 HTML/"
- The public identifier starts with: "-//Microsoft//DTD Internet Explorer 2.0 Tables/"
- The public identifier starts with: "-//Microsoft//DTD Internet Explorer 3.0 HTML Strict/"
- The public identifier starts with: "-//Microsoft//DTD Internet Explorer 3.0 HTML/"
- The public identifier starts with: "-//Microsoft//DTD Internet Explorer 3.0 Tables/"
- The public identifier starts with: "-//Netscape Comm. Corp.//DTD HTML/"
- The public identifier starts with: "-//Netscape Comm. Corp.//DTD Strict HTML/"
- The public identifier starts with: "-//O'Reilly and Associates//DTD HTML 2.0/"
- The public identifier starts with: "-//O'Reilly and Associates//DTD HTML Extended 1.0/"
- The public identifier starts with: "-//O'Reilly and Associates//DTD HTML Extended Relaxed 1.0/"
- The public identifier starts with: "-//SQ//DTD HTML 2.0 HoTMetal + extensions/"
- The public identifier starts with: "-//SoftQuad Software//DTD HoTMetal PRO"

- 6.0::19990601::extensions to HTML 4.0//"
- The public identifier starts with: "-//SoftQuad//DTD HoTMetaL PRO 4.0::19971010::extensions to HTML 4.0//"
- The public identifier starts with: "-//Spyglass//DTD HTML 2.0 Extended//"
- The public identifier starts with: "-//Sun Microsystems Corp.//DTD HotJava HTML//"
- The public identifier starts with: "-//Sun Microsystems Corp.//DTD HotJava Strict HTML//"
- The public identifier starts with: "-//W3C//DTD HTML 3 1995-03-24//"
- The public identifier starts with: "-//W3C//DTD HTML 3.2 Draft//"
- The public identifier starts with: "-//W3C//DTD HTML 3.2 Final//"
- The public identifier starts with: "-//W3C//DTD HTML 3.2//"
- The public identifier starts with: "-//W3C//DTD HTML 3.2S Draft//"
- The public identifier starts with: "-//W3C//DTD HTML 4.0 Frameset//"
- The public identifier starts with: "-//W3C//DTD HTML 4.0 Transitional//"
- The public identifier starts with: "-//W3C//DTD HTML Experimental 19960712//"
- The public identifier starts with: "-//W3C//DTD HTML Experimental 970421//"
- The public identifier starts with: "-//W3C//DTD W3 HTML//"
- The public identifier starts with: "-//W3O//DTD W3 HTML 3.0//"
- The public identifier starts with: "-//WebTechs//DTD Mozilla HTML 2.0//"
- The public identifier starts with: "-//WebTechs//DTD Mozilla HTML//"
- The system identifier is missing and the public identifier starts with: "-//W3C//DTD HTML 4.01 Frameset//"
- The system identifier is missing and the public identifier starts with: "-//W3C//DTD HTML 4.01 Transitional//"

Otherwise, if the document is *not* [an iframe srcdoc document](#), and the DOCTYPE token matches one of the conditions in the following list, then set the Document to [limited-quirks mode](#):

- The public identifier starts with: "-//W3C//DTD XHTML 1.0 Frameset//"
- The public identifier starts with: "-//W3C//DTD XHTML 1.0 Transitional//"
- The system identifier is not missing and the public identifier starts with: "-//W3C//DTD HTML 4.01 Frameset//"
- The system identifier is not missing and the public identifier starts with: "-//W3C//DTD HTML 4.01 Transitional//"

The system identifier and public identifier strings must be compared to the values given in the lists above in an [ASCII case-insensitive](#) manner. A system identifier whose value is the empty string is not considered missing for the purposes of the conditions above.

Then, switch the [insertion mode](#) to ["before html"](#).

↪ Anything else

If the document is *not* [an iframe srcdoc document](#), then this is a [parse error](#); set the Document to [quirks mode](#).

In any case, switch the [insertion mode](#) to ["before html"](#), then reprocess the token.

§ 8.2.5.4.2. THE "BEFORE HTML" INSERTION MODE

When the user agent is to apply the rules for the ["before html"](#) [insertion mode](#), the user agent must handle the token as follows:

↪ A DOCTYPE token

[parse error](#). Ignore the token.

↪ A comment token

[Insert a comment](#) as the last child of the [Document](#) object.

↪ A character token that is one of U+0009 CHARACTER TABULATION, U+000A LINE FEED (LF), U+000C FORM FEED (FF), U+000D CARRIAGE RETURN (CR), or U+0020 SPACE

Ignore the token.

↪ A start tag whose tag name is "html"

[Create an element for the token](#) in the [HTML namespace](#), with the [Document](#) as the intended parent. Append it to the [Document](#) object. Put this element in the [stack of open elements](#).

If the [Document](#) is being loaded as part of [navigation](#) of a [browsing context](#), run these steps:

1. If the result of running [match service worker registration](#) for [the Document's address](#) is non-null, run the [application cache selection algorithm](#) passing the [Document](#) object with no manifest.

2. Otherwise, run these substeps:

1. If the newly created element has a [manifest](#) attribute whose value is not the empty string, then [parse](#) the value of that attribute, relative to the newly created element, and if that is successful, run the [application cache selection algorithm](#) passing the [Document](#) object with the result of applying the [URL serializer](#) algorithm to the [resulting URL string](#) with the *exclude fragment flag* set.
2. Otherwise, run the [application cache selection algorithm](#) passing the [Document](#) object with no manifest.

Switch the [insertion mode](#) to "[before head](#)".

↪ **An end tag whose tag name is one of: "head", "body", "html", "br"**

Act as described in the "anything else" entry below.

↪ **Any other end tag**

[parse error](#). Ignore the token.

↪ **Anything else**

Create an `<html>` element whose [node document](#) is the [Document](#) object. Append it to the [Document](#) object. Put this element in the [stack of open elements](#).

If the [Document](#) is being loaded as part of [navigation](#) of a [browsing context](#), then: run the [application cache selection algorithm](#) with no manifest, passing it the [Document](#) object.

Switch the [insertion mode](#) to "[before head](#)", then reprocess the token.

The root element can end up being removed from the [Document](#) object, e.g., by scripts; nothing in particular happens in such cases, content continues being appended to the nodes as described in the next section.

§ 8.2.5.4.3. THE "BEFORE HEAD" INSERTION MODE

When the user agent is to apply the rules for the "[before head](#)" [insertion mode](#), the user agent must handle the token as follows:

↪ **A character token that is one of U+0009 CHARACTER TABULATION, U+000A LINE FEED (LF), U+000C FORM FEED (FF), U+000D CARRIAGE RETURN (CR), or U+0020 SPACE**

Ignore the token.

↪ **A comment token**

[Insert a comment](#).

↪ **A DOCTYPE token**

[parse error](#). Ignore the token.

↪ **A start tag whose tag name is "html"**

Process the token [using the rules for the "in body" insertion mode](#).

↪ **A start tag whose tag name is "head"**

[Insert an HTML element](#) for the token.

Set the [head element pointer](#) to the newly created `<head>` element.

Switch the [insertion mode](#) to "[in head](#)".

↪ **An end tag whose tag name is one of: "head", "body", "html", "br"**

Act as described in the "anything else" entry below.

↪ **Any other end tag**

[parse error](#). Ignore the token.

↪ **Anything else**

[Insert an HTML element](#) for a "head" start tag token with no attributes.

Set the [head element pointer](#) to the newly created `<head>` element.

Switch the [insertion mode](#) to "in head".

Reprocess the current token.

§ 8.2.5.4.4. THE "IN HEAD" INSERTION MODE

When the user agent is to apply the rules for the "in head" [insertion mode](#), the user agent must handle the token as follows:

- ↪ A character token that is one of U+0009 CHARACTER TABULATION, U+000A LINE FEED (LF), U+000C FORM FEED (FF), U+000D CARRIAGE RETURN (CR), or U+0020 SPACE

[Insert the character.](#)

- ↪ A comment token

[Insert a comment.](#)

- ↪ A DOCTYPE token

[parse error.](#) Ignore the token.

- ↪ A start tag whose tag name is "html"

Process the token [using the rules for](#) the "in body" [insertion mode](#).

- ↪ A start tag whose tag name is one of: "base", "basefont", "bgsound", "link"

[Insert an HTML element](#) for the token. Immediately pop the [current node](#) off the [stack of open elements](#).

[Acknowledge the token's self-closing flag](#), if it is set.

- ↪ A start tag whose tag name is "meta"

[Insert an HTML element](#) for the token. Immediately pop the [current node](#) off the [stack of open elements](#).

[Acknowledge the token's self-closing flag](#), if it is set.

If the element has a charset attribute, and [getting an encoding](#) from its value results in an [encoding](#), and the [confidence](#) is currently *tentative*, then [change the encoding](#) to the resulting encoding.

Otherwise, if the element has an http-equiv attribute whose value is an [ASCII case-insensitive](#) match for the string "Content-Type", and the element has a content attribute, and applying the [algorithm for extracting a character encoding from a meta element](#) to that attribute's value returns an [encoding](#), and the [confidence](#) is currently *tentative*, then [change the encoding](#) to the extracted encoding.

- ↪ A start tag whose tag name is "title"

Follow the [generic RCDATA element parsing algorithm](#).

- ↪ A start tag whose tag name is "noscript", if the [scripting flag](#) is enabled

- ↪ A start tag whose tag name is one of: "noframes", "style"

Follow the [generic raw text element parsing algorithm](#).

- ↪ A start tag whose tag name is "noscript", if the [scripting flag](#) is disabled

[Insert an HTML element](#) for the token.

Switch the [insertion mode](#) to "in head noscript".

- ↪ A start tag whose tag name is "script"

Run these steps:

1. Let the *adjusted insertion location* be the [appropriate place for inserting a node](#).
2. [Create an element for the token](#) in the [HTML namespace](#), with the intended parent being the element in which the *adjusted insertion location* finds itself.
3. Mark the element as being ["parser-inserted"](#) and unset the element's ["non-blocking"](#) flag.

NOTE:

This ensures that, if the script is external, any `document.write()` calls in the script will execute in-line, instead of blowing the document away, as would happen in most other cases. It also prevents the script from executing until the end tag is seen.

4. If the parser was originally created for the [HTML fragment parsing algorithm](#), then mark the `<script>` element as "already started". (fragment case)
5. Insert the newly created element at the *adjusted insertion location*.
6. Push the element onto the [stack of open elements](#) so that it is the new [current node](#).
7. Switch the tokenizer to the [§8.2.4.6 Script data state](#).
8. Let the [original insertion mode](#) be the current [insertion mode](#).
9. Switch the [insertion mode](#) to "text".

↪ **An end tag whose tag name is "head"**

Pop the [current node](#) (which will be the `<head>` element) off the [stack of open elements](#).

Switch the [insertion mode](#) to "after head".

↪ **An end tag whose tag name is one of: "body", "html", "br"**

Act as described in the "anything else" entry below.

↪ **A start tag whose tag name is "template"**

[Insert an HTML element](#) for the token.

Insert a [marker](#) at the end of the [list of active formatting elements](#).

Set the [frameset-ok flag](#) to "not ok".

Switch the [insertion mode](#) to "in template".

Push "in template" onto the [stack of template insertion modes](#) so that it is the new [current template insertion mode](#).

↪ **An end tag whose tag name is "template"**

If there is no `<template>` element on the [stack of open elements](#), then this is a [parse error](#); ignore the token.

Otherwise, run these steps:

1. [Generate all implied end tags thoroughly](#).
2. If the [current node](#) is not a `<template>` element, then this is a [parse error](#).
3. Pop elements from the [stack of open elements](#) until a `template` element has been popped from the stack.
4. [Clear the list of active formatting elements up to the last marker](#).
5. Pop the [current template insertion mode](#) off the [stack of template insertion modes](#).
6. [Reset the insertion mode appropriately](#).

↪ **A start tag whose tag name is "head"**

↪ **Any other end tag**

[parse error](#). Ignore the token.

↪ **Anything else**

Pop the [current node](#) (which will be the `<head>` element) off the [stack of open elements](#).

Switch the [insertion mode](#) to "after head".

Reprocess the token.

§ 8.2.5.4.5. THE "IN HEAD NOSCRIPT" INSERTION MODE

When the user agent is to apply the rules for the ["in head noscript" insertion mode](#), the user agent must handle the token as follows:

↪ **A DOCTYPE token**

[parse error](#). Ignore the token.

↪ **A start tag whose tag name is "html"**

Process the token [using the rules for the "in body" insertion mode](#).

↪ **An end tag whose tag name is "noscript"**

Pop the [current node](#) (which will be a [<noscript>](#) element) from the [stack of open elements](#); the new [current node](#) will be a [<head>](#) element.

Switch the [insertion mode](#) to ["in head"](#).

↪ **A character token that is one of U+0009 CHARACTER TABULATION, U+000A LINE FEED (LF), U+000C FORM FEED (FF), U+000D CARRIAGE RETURN (CR), or U+0020 SPACE**

↪ **A comment token**

↪ **A start tag whose tag name is one of: "basefont", "bgsound", "link", "meta", "noframes", "style"**

Process the token [using the rules for the "in head" insertion mode](#).

↪ **An end tag whose tag name is "br"**

Act as described in the "anything else" entry below.

↪ **A start tag whose tag name is one of: "head", "noscript"**

↪ **Any other end tag**

[parse error](#). Ignore the token.

↪ **Anything else**

[parse error](#).

Pop the [current node](#) (which will be a [<noscript>](#) element) from the [stack of open elements](#); the new [current node](#) will be a [<head>](#) element.

Switch the [insertion mode](#) to ["in head"](#).

Reprocess the token.

§ 8.2.5.4.6. THE "AFTER HEAD" INSERTION MODE

When the user agent is to apply the rules for the ["after head" insertion mode](#), the user agent must handle the token as follows:

↪ **A character token that is one of U+0009 CHARACTER TABULATION, U+000A LINE FEED (LF), U+000C FORM FEED (FF), U+000D CARRIAGE RETURN (CR), or U+0020 SPACE**

[Insert the character](#).

↪ **A comment token**

[Insert a comment](#).

↪ **A DOCTYPE token**

[parse error](#). Ignore the token.

↪ **A start tag whose tag name is "html"**

Process the token [using the rules for the "in body" insertion mode](#).

↪ **A start tag whose tag name is "body"**

[Insert an HTML element](#) for the token.

Set the [frameset-ok flag](#) to "not ok".

Switch the [insertion mode](#) to "[in body](#)".

↪ **A start tag whose tag name is "frameset"**

[Insert an HTML element](#) for the token.

Switch the [insertion mode](#) to "[in frameset](#)".

↪ **A start tag whose tag name is one of: "base", "basefont", "bgsound", "link", "meta", "noframes", "script", "style", "template", "title"**

[parse error](#).

Push the node pointed to by the [head element pointer](#) onto the [stack of open elements](#).

Process the token [using the rules for](#) the "[in head](#)" [insertion mode](#).

Remove the node pointed to by the [head element pointer](#) from the [stack of open elements](#). (It might not be the [current node](#) at this point.)

NOTE:

The [head element pointer](#) cannot be null at this point.

↪ **An end tag whose tag name is "template"**

Process the token [using the rules for](#) the "[in head](#)" [insertion mode](#).

↪ **An end tag whose tag name is one of: "body", "html", "br"**

Act as described in the "anything else" entry below.

↪ **A start tag whose tag name is "head"**

↪ **Any other end tag**

[parse error](#). Ignore the token.

↪ **Anything else**

[Insert an HTML element](#) for a "body" start tag token with no attributes.

Switch the [insertion mode](#) to "[in body](#)".

Reprocess the current token.

§ 8.2.5.4.7. THE "IN BODY" INSERTION MODE

When the user agent is to apply the rules for the "[in body](#)" [insertion mode](#), the user agent must handle the token as follows:

↪ **A character token that is U+0000 NULL**

[parse error](#). Ignore the token.

↪ **A character token that is one of U+0009 CHARACTER TABULATION, U+000A LINE FEED (LF), U+000C FORM FEED (FF), U+000D CARRIAGE RETURN (CR), or U+0020 SPACE**

[reconstruct the active formatting elements](#), if any.

[Insert the token's character](#).

↪ **Any other character token**

[reconstruct the active formatting elements](#), if any.

[Insert the token's character](#).

Set the [frameset-ok flag](#) to "not ok".

↪ **A comment token**

[Insert a comment](#).

↪ **A DOCTYPE token**

[parse error](#). Ignore the token.

↪ **A start tag whose tag name is "html"**

[parse error](#).

If there is a `<template>` element on the [stack of open elements](#), then ignore the token.

Otherwise, for each attribute on the token, check to see if the attribute is already present on the top element of the [stack of open elements](#). If it is not, add the attribute and its corresponding value to that element.

↪ **A start tag whose tag name is one of: "base", "basefont", "bgsound", "link", "meta", "noframes", "script", "style", "template", "title"**

↪ **An end tag whose tag name is "template"**

Process the token [using the rules for](#) the "in head" [insertion mode](#).

↪ **A start tag whose tag name is "body"**

[parse error](#).

If the second element on the [stack of open elements](#) is not a body element, if the [stack of open elements](#) has only one node on it, or if there is a `<template>` element on the [stack of open elements](#), then ignore the token. ([fragment case](#))

Otherwise, set the [frameset-ok flag](#) to "not ok"; then, for each attribute on the token, check to see if the attribute is already present on the `<body>` element (the second element) on the [stack of open elements](#), and if it is not, add the attribute and its corresponding value to that element.

↪ **A start tag whose tag name is "frameset"**

[parse error](#).

If the [stack of open elements](#) has only one node on it, or if the second element on the [stack of open elements](#) is not a `<body>` element, then ignore the token. ([fragment case](#))

If the [frameset-ok flag](#) is set to "not ok", ignore the token.

Otherwise, run the following steps:

1. Remove the second element on the [stack of open elements](#) from its parent node, if it has one.
2. Pop all the nodes from the bottom of the [stack of open elements](#), from the [current node](#) up to, but not including, the root `<html>` element.
3. [Insert an HTML element](#) for the token.
4. Switch the [insertion mode](#) to "in frameset".

↪ **An end-of-file token**

If the [stack of template insertion modes](#) is not empty, then process the token [using the rules for](#) the "in template" [insertion mode](#).

Otherwise, follow these steps:

1. If there is a node in the [stack of open elements](#) that is not either a `<dd>` element, a `<dt>` element, an `<li>` element, an `<optgroup>` element, an `<option>` element, a `<p>` element, an `<rb>` element, an `<rp>` element, an `<rt>` element, an `<rtc>` element, a `<tbody>` element, a `<td>` element, a `<tfoot>` element, a `<th>` element, a `<thead>` element, a `<tr>` element, the `<body>` element, or the `<html>` element, then this is a [parse error](#).
2. [Stop parsing](#).

↪ **An end tag whose tag name is "body"**

If the [stack of open elements](#) does not [have a body element in scope](#), this is a [parse error](#); ignore the token.

Otherwise, if there is a node in the [stack of open elements](#) that is not either a `<dd>` element, a `<dt>` element, an `<li>` element, an `<optgroup>` element, an `<option>` element, a `<p>` element, an `<rb>` element, an `<rp>` element, an `<rt>` element, an `<rtc>` element, a `<tbody>` element, a `<td>` element, a `<tfoot>` element, a `<th>` element, a `<thead>` ele-

ment, a [<tr>](#) element, the [<body>](#) element, or the [<html>](#) element, then this is a [parse error](#).

Switch the [insertion mode](#) to "after body".

↪ An end tag whose tag name is "html"

If the [stack of open elements](#) does not [have a body element in scope](#), this is a [parse error](#); ignore the token.

Otherwise, if there is a node in the [stack of open elements](#) that is not either a [<dd>](#) element, a [<dt>](#) element, an [](#) element, an [<optgroup>](#) element, an [<option>](#) element, a [<p>](#) element, an [<rb>](#) element, an [<rp>](#) element, an [<rtc>](#) element, an [<rt>](#) element, a [<tbody>](#) element, a [<td>](#) element, a [<tfoot>](#) element, a [<th>](#) element, a [<thead>](#) element, a [<tr>](#) element, the [<body>](#) element, or the [<html>](#) element, then this is a [parse error](#).

Switch the [insertion mode](#) to "after body".

Reprocess the token.

↪ A start tag whose tag name is one of: "address", "article", "aside", "blockquote", "center", "details", "div", "dl", "fieldset", "figcaption", "figure", "footer", "header", "main", "menu", "nav", "ol", "p", "section", "summary", "ul"

If the [stack of open elements](#) [has a p element in button scope](#), then [close a p element](#).

[Insert an HTML element](#) for the token.

↪ A start tag whose tag name is one of: "h1", "h2", "h3", "h4", "h5", "h6"

If the [stack of open elements](#) [has a p element in button scope](#), then [close a p element](#).

If the [current node](#) is an [HTML element](#) whose tag name is one of "h1", "h2", "h3", "h4", "h5", or "h6", then this is a [parse error](#); pop the [current node](#) off the [stack of open elements](#).

[Insert an HTML element](#) for the token.

↪ A start tag whose tag name is one of: "pre", "listing"

If the [stack of open elements](#) [has a p element in button scope](#), then [close a p element](#).

[Insert an HTML element](#) for the token.

If the [next token](#) is a U+000A LINE FEED (LF) character token, then ignore that token and move on to the next one. (Newlines at the start of pre blocks are ignored as an authoring convenience.)

Set the [frameset-ok flag](#) to "not ok".

↪ A start tag whose tag name is "form"

If the [form element pointer](#) is not null, and there is no [<template>](#) element on the [stack of open elements](#), then this is a [parse error](#); ignore the token.

Otherwise:

If the [stack of open elements](#) [has a p element in button scope](#), then [close a p element](#).

[Insert an HTML element](#) for the token, and, if there is no [template](#) element on the [stack of open elements](#), set the [form element pointer](#) to point to the element created.

↪ A start tag whose tag name is "li"

Run these steps:

1. Set the [frameset-ok flag](#) to "not ok".
2. Initialize *node* to be the [current node](#) (the bottommost node of the stack).
3. *Loop*: If *node* is an [](#) element, then run these substeps:
 1. [Generate implied end tags](#), except for [](#) elements.
 2. If the [current node](#) is not an [](#) element, then this is a [parse error](#).
 3. Pop elements from the [stack of open elements](#) until an [li](#) element has been popped from the stack.

4. Jump to the step labeled *done* below.
4. If *node* is in the [special](#) category, but is not an address, div, or [<p>](#) element, then jump to the step labeled *done* below.
5. Otherwise, set *node* to the previous entry in the [stack of open elements](#) and return to the step labeled *loop*.
6. *Done*: If the [stack of open elements](#) has a [<p>](#) element in button scope, then close a [<p>](#) element.
7. Finally, [insert an HTML element](#) for the token.

↪ **A start tag whose tag name is one of: "dd", "dt"**

Run these steps:

1. Set the [frameset-ok flag](#) to "not ok".
2. Initialize *node* to be the [current node](#) (the bottommost node of the stack).
3. *Loop*: If *node* is a [<dd>](#) element, then run these substeps:
 1. [Generate implied end tags](#), except for [<dd>](#) elements.
 2. If the [current node](#) is not a [<dd>](#) element, then this is a [parse error](#).
 3. Pop elements from the [stack of open elements](#) until a dd element has been popped from the stack.
 4. Jump to the step labeled *done* below.
4. If *node* is a [<dt>](#) element, then run these substeps:
 1. [Generate implied end tags](#), except for [<dt>](#) elements.
 2. If the [current node](#) is not a [<dt>](#) element, then this is a [parse error](#).
 3. Pop elements from the [stack of open elements](#) until a dt element has been popped from the stack.
 4. Jump to the step labeled *done* below.
5. If *node* is in the [special](#) category, but is not an address, div, or [<p>](#) element, then jump to the step labeled *done* below.
6. Otherwise, set *node* to the previous entry in the [stack of open elements](#) and return to the step labeled *loop*.
7. *Done*: If the [stack of open elements](#) has a p element in button scope, then [close a p element](#).
8. Finally, [insert an HTML element](#) for the token.

↪ **A start tag whose tag name is "plaintext"**

If the [stack of open elements](#) has a p element in button scope, then [close a p element](#).

[Insert an HTML element](#) for the token.

Switch the tokenizer to the [§8.2.4.7 PLAINTEXT](#) state.

NOTE:

Once a start tag with the tag name "plaintext" has been seen, that will be the last token ever seen other than character tokens (and the end-of-file token), because there is no way to switch out of the [§8.2.4.7 PLAINTEXT](#) state.

↪ **A start tag whose tag name is "button"**

1. If the [stack of open elements](#) has a button element in scope, then run these substeps:
 1. [parse error](#).
 2. [Generate implied end tags](#).

3. Pop elements from the [stack of open elements](#) until a button element has been popped from the stack.

2. [reconstruct the active formatting elements](#), if any.

3. [Insert an HTML element](#) for the token.

4. Set the [frameset-ok flag](#) to "not ok".

↪ **An end tag whose tag name is one of: "address", "article", "aside", "blockquote", "button", "center", "details", "dir", "div", "dl", "fieldset", "figcaption", "figure", "footer", "header", "listing", "main", "menu", "nav", "ol", "pre", "section", "summary", "ul"**

If the [stack of open elements](#) does not [have an element in scope](#) that is an [HTML element](#) with the same tag name as that of the token, then this is a [parse error](#); ignore the token.

Otherwise, run these steps:

1. [Generate implied end tags](#).

2. If the [current node](#) is not an [HTML element](#) with the same tag name as that of the token, then this is a [parse error](#).

3. Pop elements from the [stack of open elements](#) until an [HTML element](#) with the same tag name as the token has been popped from the stack.

↪ **An end tag whose tag name is "form"**

If there is no [<template>](#) element on the [stack of open elements](#), then run these substeps:

1. Let [node](#) be the element that the [form element pointer](#) is set to, or null if it is not set to an element.

2. Set the [form element pointer](#) to null.

3. If [node](#) is null or if the [stack of open elements](#) does not [have node in scope](#), then this is a [parse error](#); abort these steps and ignore the token.

4. [Generate implied end tags](#).

5. If the [current node](#) is not [node](#), then this is a [parse error](#).

6. Remove [node](#) from the [stack of open elements](#).

If there is a [<template>](#) element on the [stack of open elements](#), then run these substeps instead:

1. If the [stack of open elements](#) does not [have a form element in scope](#), then this is a [parse error](#); abort these steps and ignore the token.

2. [Generate implied end tags](#).

3. If the [current node](#) is not a [<form>](#) element, then this is a [parse error](#).

4. Pop elements from the [stack of open elements](#) until a form element has been popped from the stack.

↪ **An end tag whose tag name is "p"**

If the [stack of open elements](#) does not [have a p element in button scope](#), then this is a [parse error](#); [insert an HTML element](#) for a "p" start tag token with no attributes.

[Close a p element](#).

↪ **An end tag whose tag name is "li"**

If the [stack of open elements](#) does not [have an li element in list item scope](#), then this is a [parse error](#); ignore the token.

Otherwise, run these steps:

1. [Generate implied end tags](#), except for [](#) elements.

2. If the [current node](#) is not an [](#) element, then this is a [parse error](#).

3. Pop elements from the [stack of open elements](#) until an `li` element has been popped from the stack.

↪ **An end tag whose tag name is one of: "dd", "dt"**

If the [stack of open elements](#) does not [have an element in scope](#) that is an [HTML element](#) with the same tag name as that of the token, then this is a [parse error](#); ignore the token.

Otherwise, run these steps:

1. [Generate implied end tags](#), except for [html elements](#) with the same tag name as the token.
2. If the [current node](#) is not an [HTML element](#) with the same tag name as that of the token, then this is a [parse error](#).
3. Pop elements from the [stack of open elements](#) until an [HTML element](#) with the same tag name as the token has been popped from the stack.

↪ **An end tag whose tag name is one of: "h1", "h2", "h3", "h4", "h5", "h6"**

If the [stack of open elements](#) does not [have an element in scope](#) that is an [HTML element](#) and whose tag name is one of "h1", "h2", "h3", "h4", "h5", or "h6", then this is a [parse error](#); ignore the token.

Otherwise, run these steps:

1. [Generate implied end tags](#).
2. If the [current node](#) is not an [HTML element](#) with the same tag name as that of the token, then this is a [parse error](#).
3. Pop elements from the [stack of open elements](#) until an [HTML element](#) whose tag name is one of "h1", "h2", "h3", "h4", "h5", or "h6" has been popped from the stack.

↪ **An end tag whose tag name is "sarcasm"**

Take a deep breath, then act as described in the "any other end tag" entry below.

↪ **A start tag whose tag name is "a"**

If the [list of active formatting elements](#) contains an `<a>` element between the end of the list and the last [marker](#) on the list (or the start of the list if there is no [marker](#) on the list), then this is a [parse error](#); run the [adoption agency algorithm](#) for the tag name "a", then remove that element from the [list of active formatting elements](#) and the [stack of open elements](#) if the [adoption agency algorithm](#) didn't already remove it (it might not have if the element is not [in table scope](#)).

EXAMPLE 649

In the non-conforming stream `<a href="a">a<table><a href="b">b</table>x`, the first `<a>` element would be closed upon seeing the second one, and the "x" character would be inside a link to "b", not to "a". This is despite the fact that the outer `<a>` element is not in table scope (meaning that a regular `</a>` end tag at the start of the table wouldn't close the outer `<a>` element). The result is that the two `<a>` elements are indirectly nested inside each other — non-conforming markup will often result in non-conforming DOMs when parsed.

[reconstruct the active formatting elements](#), if any.

[Insert an HTML element](#) for the token. [Push onto the list of active formatting elements](#) that element.

↪ **A start tag whose tag name is one of: "b", "big", "code", "em", "font", "i", "s", "small", "strike", "strong", "tt", "u"**

[reconstruct the active formatting elements](#), if any.

[Insert an HTML element](#) for the token. [Push onto the list of active formatting elements](#) that element.

↪ **A start tag whose tag name is "nobr"**

[reconstruct the active formatting elements](#), if any.

If the [stack of open elements](#) has a `nobr` element in scope, then this is a [parse error](#); run the [adoption agency algorithm](#) for the tag name "nobr", then once again [reconstruct the active formatting elements](#), if any.

[Insert an HTML element](#) for the token. [Push onto the list of active formatting elements](#) that element.

- ↪ An end tag whose tag name is one of: "a", "b", "big", "code", "em", "font", "i", "nobr", "s", "small", "strike", "strong", "tt", "u"

Run the [adoption agency algorithm](#) for the token's tag name.

- ↪ A start tag whose tag name is one of: "applet", "marquee", "object"

[reconstruct the active formatting elements](#), if any.

[Insert an HTML element](#) for the token.

Insert a [marker](#) at the end of the [list of active formatting elements](#).

Set the [frameset-ok flag](#) to "not ok".

- ↪ An end tag token whose tag name is one of: "applet", "marquee", "object"

If the [stack of open elements](#) does not [have an element in scope](#) that is an [HTML element](#) with the same tag name as that of the token, then this is a [parse error](#); ignore the token.

Otherwise, run these steps:

1. [Generate implied end tags](#).
2. If the [current node](#) is not an [HTML element](#) with the same tag name as that of the token, then this is a [parse error](#).
3. Pop elements from the [stack of open elements](#) until an [HTML element](#) with the same tag name as the token has been popped from the stack.
4. [Clear the list of active formatting elements up to the last marker](#).

- ↪ A start tag whose tag name is "table"

If the Document is *not* set to [quirks mode](#), and the [stack of open elements](#) has a [p element in button scope](#), then [close a p element](#).

[Insert an HTML element](#) for the token.

Set the [frameset-ok flag](#) to "not ok".

Switch the [insertion mode](#) to "in table".

- ↪ An end tag whose tag name is "br"

[parse error](#). Drop the attributes from the token, and act as described in the next entry; i.e., act as if this was a "br" start tag token with no attributes, rather than the end tag token that it actually is.

- ↪ A start tag whose tag name is one of: "area", "br", "embed", "img", "keygen", "wbr"

[reconstruct the active formatting elements](#), if any.

[Insert an HTML element](#) for the token. Immediately pop the [current node](#) off the [stack of open elements](#).

[Acknowledge the token's self-closing flag](#), if it is set.

Set the [frameset-ok flag](#) to "not ok".

- ↪ A start tag whose tag name is "input"

[reconstruct the active formatting elements](#), if any.

[Insert an HTML element](#) for the token. Immediately pop the [current node](#) off the [stack of open elements](#).

[Acknowledge the token's self-closing flag](#), if it is set.

If the token does not have an attribute with the name "type", or if it does, but that attribute's value is not an [ASCII case-insensitive](#) match for the string "hidden", then: set the [frameset-ok flag](#) to "not ok".

- ↪ A start tag whose tag name is one of: "menuitem", "param", "source", "track"

[Insert an HTML element](#) for the token. Immediately pop the [current node](#) off the [stack of open elements](#).

[Acknowledge the token's self-closing flag](#), if it is set.

↪ **A start tag whose tag name is "hr"**

If the [stack of open elements](#) has a [p element in button scope](#), then [close a p element](#).

[Insert an HTML element](#) for the token. Immediately pop the [current node](#) off the [stack of open elements](#).

[Acknowledge the token's self-closing flag](#), if it is set.

Set the [frameset-ok flag](#) to "not ok".

↪ **A start tag whose tag name is "image"**

[parse error](#). Change the token's tag name to "img" and reprocess it. (Don't ask.)

↪ **A start tag whose tag name is "textarea"**

Run these steps:

1. [Insert an HTML element](#) for the token.
2. If the [next token](#) is a U+000A LINE FEED (LF) character token, then ignore that token and move on to the next one. (Newlines at the start of textarea elements are ignored as an authoring convenience.)
3. Switch the tokenizer to the [§8.2.4.3 RCDATA state](#).
4. Let the [original insertion mode](#) be the current [insertion mode](#).
5. Set the [frameset-ok flag](#) to "not ok".
6. Switch the [insertion mode](#) to "text".

↪ **A start tag whose tag name is "xmp"**

If the [stack of open elements](#) has a [p element in button scope](#), then [close a p element](#).

[reconstruct the active formatting elements](#), if any.

Set the [frameset-ok flag](#) to "not ok".

Follow the [generic raw text element parsing algorithm](#).

↪ **A start tag whose tag name is "iframe"**

Set the [frameset-ok flag](#) to "not ok".

Follow the [generic raw text element parsing algorithm](#).

↪ **A start tag whose tag name is "noembed"**

↪ **A start tag whose tag name is "noscript", if the [scripting flag](#) is enabled**

Follow the [generic raw text element parsing algorithm](#).

↪ **A start tag whose tag name is "select"**

[reconstruct the active formatting elements](#), if any.

[Insert an HTML element](#) for the token.

Set the [frameset-ok flag](#) to "not ok".

If the [insertion mode](#) is one of "in table", "in caption", "in table body", "in row", or "in cell", then switch the [insertion mode](#) to "in select in table". Otherwise, switch the [insertion mode](#) to "in select".

↪ **A start tag whose tag name is one of: "optgroup", "option"**

If the [current node](#) is an `<option>` element, then pop the [current node](#) off the [stack of open elements](#).

[reconstruct the active formatting elements](#), if any.

[Insert an HTML element](#) for the token.

↪ A start tag whose tag name is one of: "rb", "rtc"

If the [stack of open elements](#) has a [ruby element in scope](#), then [generate implied end tags](#). If the [current node](#) is not now a [<ruby> element](#), this is a [parse error](#).
[Insert an HTML element](#) for the token.

↪ A start tag whose tag name is one of: "rp", "rt"

If the [stack of open elements](#) has a [ruby element in scope](#), then [generate implied end tags](#), except for [<rtc> elements](#). If the [current node](#) is not then a ruby element or an [<rtc> element](#), this is a [parse error](#).

[Insert an HTML element](#) for the token.

↪ A start tag whose tag name is "math"

[reconstruct the active formatting elements](#), if any.

[Adjust MathML attributes](#) for the token. (This fixes the case of MathML attributes that are not all lowercase.)

[Adjust foreign attributes](#) for the token. (This fixes the use of namespaced attributes, in particular XLink.)

[Insert a foreign element](#) for the token, in the [MathML namespace](#).

If the token has its *self-closing flag* set, pop the [current node](#) off the [stack of open elements](#) and [acknowledge the token's self-closing flag](#).

↪ A start tag whose tag name is "svg"

[reconstruct the active formatting elements](#), if any.

[Adjust SVG attributes](#) for the token. (This fixes the case of SVG attributes that are not all lowercase.)

[Adjust foreign attributes](#) for the token. (This fixes the use of namespaced attributes, in particular XLink in SVG.)

[Insert a foreign element](#) for the token, in the [SVG namespace](#).

If the token has its *self-closing flag* set, pop the [current node](#) off the [stack of open elements](#) and [acknowledge the token's self-closing flag](#).

↪ A start tag whose tag name is one of: "caption", "col", "colgroup", "frame", "head", "tbody", "td", "tfoot", "th", "thead", "tr"

[parse error](#). Ignore the token.

↪ Any other start tag

[reconstruct the active formatting elements](#), if any.

[Insert an HTML element](#) for the token.

NOTE:

This element will be an [ordinary](#) element.

↪ Any other end tag

Run these steps:

1. Initialize *node* to be the [current node](#) (the bottommost node of the stack).
2. *Loop*: If *node* is an [HTML element](#) with the same tag name as the token, then:
 1. [Generate implied end tags](#), except for [html elements](#) with the same tag name as the token.
 2. If *node* is not the [current node](#), then this is a [parse error](#).
 3. Pop all the nodes from the [current node](#) up to *node*, including *node*, then stop these steps.
3. Otherwise, if *node* is in the [special](#) category, then this is a [parse error](#); ignore the token, and abort these steps.
4. Set *node* to the previous entry in the [stack of open elements](#).
5. Return to the step labeled *loop*.

When the steps above say the user agent is to **close a `<p>` element**, it means that the user agent must run the following steps:

1. [Generate implied end tags](#), except for `<p>` elements.
2. If the [current node](#) is not a `<p>` element, then this is a [parse error](#).
3. Pop elements from the [stack of open elements](#) until a `<p>` element has been popped from the stack.

The **adoption agency algorithm**, which takes as its only argument a tag name `subject` for which the algorithm is being run, consists of the following steps:

1. If the [current node](#) is an [HTML element](#) whose tag name is `subject`, and the [current node](#) is not in the [list of active formatting elements](#), then pop the [current node](#) off the [stack of open elements](#), and abort these steps.
2. Let `outer loop counter` be zero.
3. *Outer loop*: If `outer loop counter` is greater than or equal to eight, then abort these steps.
4. Increment `outer loop counter` by one.
5. Let `formatting element` be the last element in the [list of active formatting elements](#) that:
 - is between the end of the list and the last [marker](#) in the list, if any, or the start of the list otherwise, and
 - has the tag name `subject`.

If there is no such element, then abort these steps and instead act as described in the "any other end tag" entry above.

6. If `formatting element` is not in the [stack of open elements](#), then this is a [parse error](#); remove the element from the list, and abort these steps.
7. If `formatting element` is in the [stack of open elements](#), but the element is not [in scope](#), then this is a [parse error](#); abort these steps.
8. If `formatting element` is not the [current node](#), this is a [parse error](#). (But do not abort these steps.)
9. Let `furthest block` be the topmost node in the [stack of open elements](#) that is lower in the stack than `formatting element`, and is an element in the [special](#) category. There might not be one.
10. If there is no `furthest block`, then the user agent must first pop all the nodes from the bottom of the [stack of open elements](#), from the [current node](#) up to and including `formatting element`, then remove `formatting element` from the [list of active formatting elements](#), and finally abort these steps.
11. Let `common ancestor` be the element immediately above `formatting element` in the [stack of open elements](#).
12. Let a bookmark note the position of `formatting element` in the [list of active formatting elements](#) relative to the elements on either side of it in the list.
13. Let `node` and `last node` be `furthest block`. Follow these steps:
 1. Let `inner loop counter` be zero.
 2. *Inner loop*: Increment `inner loop counter` by one.
 3. Let `node` be the element immediately above `node` in the [stack of open elements](#), or if `node` is no longer in the [stack of open elements](#) (e.g., because it got removed by this algorithm), the element that was immediately above `node` in the [stack of open elements](#) before `node` was removed.
 4. If `node` is `formatting element`, then go to the next step in the overall algorithm.
 5. If `inner loop counter` is greater than three and `node` is in the [list of active formatting elements](#), then remove `node` from the [list of active formatting elements](#).
 6. If `node` is not in the [list of active formatting elements](#), then remove `node` from the [stack of open elements](#) and then go back to the step labeled *inner loop*.
 7. [Create an element for the token](#) for which the element `node` was created, in the [HTML namespace](#), with `common`

- ancestor* as the intended parent; replace the entry for *node* in the [list of active formatting elements](#) with an entry for the new element, replace the entry for *node* in the [stack of open elements](#) with an entry for the new element, and let *node* be the new element.
8. If *last node* is *furthest block*, then move the aforementioned bookmark to be immediately after the new *node* in the [list of active formatting elements](#).
 9. Insert *last node* into *node*, first removing it from its previous parent node if any.
 10. Let *last node* be *node*.
 11. Return to the step labeled *inner loop*.
 14. Insert whatever *last node* ended up being in the previous step at the [appropriate place for inserting a node](#), but using *common ancestor* as the *override target*.
 15. [Create an element for the token](#) for which *formatting element* was created, in the [HTML namespace](#), with *furthest block* as the intended parent.
 16. Take all of the child nodes of *furthest block* and append them to the element created in the last step.
 17. Append that new element to *furthest block*.
 18. Remove *formatting element* from the [list of active formatting elements](#), and insert the new element into the [list of active formatting elements](#) at the position of the aforementioned bookmark.
 19. Remove *formatting element* from the [stack of open elements](#), and insert the new element into the [stack of open elements](#) immediately below the position of *furthest block* in that stack.
 20. Jump back to the step labeled *outer loop*.

NOTE:

This algorithm's name, the "adoption agency algorithm", comes from the way it causes elements to change parents, and is in contrast with other possible algorithms for dealing with misnested content, which included the "incest algorithm", the "secret affair algorithm", and the "Heisenberg algorithm".

§ 8.2.5.4.8. THE "TEXT" INSERTION MODE

When the user agent is to apply the rules for the ["text" insertion mode](#), the user agent must handle the token as follows:

↪ A character token

[Insert the token's character.](#)

NOTE:

This can never be a U+0000 NULL character; the tokenizer converts those to U+FFFD REPLACEMENT CHARACTER characters.

↪ An end-of-file token

[parse error.](#)

If the [current node](#) is a [<script>](#) element, mark the [<script>](#) element as ["already started"](#).

Pop the [current node](#) off the [stack of open elements](#).

Switch the [insertion mode](#) to the [original insertion mode](#) and reprocess the token.

↪ An end tag whose tag name is "script"

If the [JavaScript execution context stack](#) is empty, [perform a microtask checkpoint](#).

Let *script* be the [current node](#) (which will be a [<script>](#) element).

Pop the [current node](#) off the [stack of open elements](#).

Switch the [insertion mode](#) to the [original insertion mode](#).

Let the [old insertion point](#) have the same value as the current [insertion point](#). Let the [insertion point](#) be just before the [next input character](#).

Increment the parser's [script nesting level](#) by one.

Prepare the [script](#). This might cause some script to execute, which might cause [new characters to be inserted into the tokenizer](#), and might cause the tokenizer to output more tokens, resulting in a [reentrant invocation of the parser](#).

Decrement the parser's [script nesting level](#) by one. If the parser's [script nesting level](#) is zero, then set the [parser pause flag](#) to false.

Let the [insertion point](#) have the value of the [old insertion point](#). (In other words, restore the [insertion point](#) to its previous value. This value might be the "undefined" value.)

At this stage, if there is a [pending parsing-blocking script](#), then:

↪ **If the [script nesting level](#) is not zero:**

Set the [parser pause flag](#) to true, and abort the processing of any nested invocations of the tokenizer, yielding control back to the caller. (Tokenization will resume when the caller returns to the "outer" tree construction stage.)

NOTE:

The tree construction stage of this particular parser is [being called reentrantly](#), say from a call to `document.write()`.

↪ **Otherwise:**

Run these steps:

1. Let *the script* be the [pending parsing-blocking script](#). There is no longer a [pending parsing-blocking script](#).
2. Block the [tokenizer](#) for this instance of the [HTML parser](#), such that the [event loop](#) will not run [tasks](#) that invoke the [tokenizer](#).
3. If the parser's Document [has a style sheet that is blocking scripts](#) or *the script*'s "ready to be parser-executed" flag is not set: [spin the event loop](#) until the parser's Document [has no style sheet that is blocking scripts](#) and *the script*'s "ready to be parser-executed" flag is set.
4. If this [parser has been aborted](#) in the meantime, abort these steps.

NOTE:

This could happen if, e.g., while the [spin the event loop](#) algorithm is running, the [browsing context](#) gets closed, or the `document.open()` method gets invoked on the [Document](#).

5. Unblock the [tokenizer](#) for this instance of the [HTML parser](#), such that [tasks](#) that invoke the [tokenizer](#) can again be run.
6. Let the [insertion point](#) be just before the [next input character](#).
7. Increment the parser's [script nesting level](#) by one (it should be zero before this step, so this sets it to one).
8. [Execute](#) *the script*.
9. Decrement the parser's [script nesting level](#) by one. If the parser's [script nesting level](#) is zero (which it always should be at this point), then set the [parser pause flag](#) to false.
10. Let the [insertion point](#) be undefined again.
11. If there is once again a [pending parsing-blocking script](#), then repeat these steps from step 1.

↪ **Any other end tag**

Pop the [current node](#) off the [stack of open elements](#).

Switch the [insertion mode](#) to the [original insertion mode](#).

§ 8.2.5.4.9. THE "IN TABLE" INSERTION MODE

When the user agent is to apply the rules for the ["in table" insertion mode](#), the user agent must handle the token as follows:

↪ **A character token, if the [current node](#) is `<table>`, `<tbody>`, `<tfoot>`, `<thead>`, or `<tr>` element**

Let the [pending table character tokens](#) be an empty list of tokens.

Let the [original insertion mode](#) be the current [insertion mode](#).

Switch the [insertion mode](#) to ["in table text"](#) and reprocess the token.

↪ **A comment token**

[Insert a comment](#).

↪ **A DOCTYPE token**

[parse error](#). Ignore the token.

↪ **A start tag whose tag name is "caption"**

[Clear the stack back to a table context](#). (See below.)

Insert a [marker](#) at the end of the [list of active formatting elements](#).

[Insert an HTML element](#) for the token, then switch the [insertion mode](#) to ["in caption"](#).

↪ **A start tag whose tag name is "colgroup"**

[Clear the stack back to a table context](#). (See below.)

[Insert an HTML element](#) for the token, then switch the [insertion mode](#) to ["in column group"](#).

↪ **A start tag whose tag name is "col"**

[Clear the stack back to a table context](#). (See below.)

[Insert an HTML element](#) for a "colgroup" start tag token with no attributes, then switch the [insertion mode](#) to ["in column group"](#).

Reprocess the current token.

↪ **A start tag whose tag name is one of: "tbody", "tfoot", "thead"**

[Clear the stack back to a table context](#). (See below.)

[Insert an HTML element](#) for the token, then switch the [insertion mode](#) to ["in table body"](#).

↪ **A start tag whose tag name is one of: "td", "th", "tr"**

[Clear the stack back to a table context](#). (See below.)

[Insert an HTML element](#) for a "tbody" start tag token with no attributes, then switch the [insertion mode](#) to ["in table body"](#).

Reprocess the current token.

↪ **A start tag whose tag name is "table"**

[parse error](#).

If the [stack of open elements](#) does not [have a table element in table scope](#), ignore the token.

Otherwise:

Pop elements from this stack until a [table element](#) has been popped from the stack.

[Reset the insertion mode appropriately.](#)

Reprocess the token.

↪ **An end tag whose tag name is "table"**

If the [stack of open elements](#) does not [have a table element in table scope](#), this is a [parse error](#); ignore the token.

Otherwise:

Pop elements from this stack until a [<table>](#) element has been popped from the stack.

[Reset the insertion mode appropriately.](#)

↪ **An end tag whose tag name is one of: "body", "caption", "col", "colgroup", "html", "tbody", "td", "tfoot", "th", "thead", "tr"**

[parse error](#). Ignore the token.

↪ **A start tag whose tag name is one of: "style", "script", "template"**

↪ **An end tag whose tag name is "template"**

Process the token [using the rules for the "in head" insertion mode](#).

↪ **A start tag whose tag name is "input"**

If the token does not have an attribute with the name "type", or if it does, but that attribute's value is not an [ASCII case-insensitive](#) match for the string "hidden", then: act as described in the "anything else" entry below.

Otherwise:

[parse error](#).

[Insert an HTML element](#) for the token.

Pop that [<input>](#) element off the [stack of open elements](#).

[Acknowledge the token's self-closing flag](#), if it is set.

↪ **A start tag whose tag name is "form"**

[parse error](#).

If there is a [<template>](#) element on the [stack of open elements](#), or if the [form element pointer](#) is not null, ignore the token.

Otherwise:

[Insert an HTML element](#) for the token, and set the [form element pointer](#) to point to the element created.

Pop that [<form>](#) element off the [stack of open elements](#).

↪ **An end-of-file token**

Process the token [using the rules for the "in body" insertion mode](#).

↪ **Anything else**

[parse error](#). Enable [foster parenting](#), process the token [using the rules for the "in body" insertion mode](#), and then disable [foster parenting](#).

When the steps above require the user agent to **clear the stack back to a table context**, it means that the user agent must, while the [current node](#) is not a [<table>](#), [<template>](#), or [<html>](#) element, pop elements from the [stack of open elements](#).

NOTE:

This is the same list of elements as used in the [has an element in table scope](#) steps.

NOTE:

The [current node](#) being an [<html>](#) element after this process is a [fragment case](#).

§ 8.2.5.4.10. THE "IN TABLE TEXT" INSERTION MODE

When the user agent is to apply the rules for the "[in table text](#)" [insertion mode](#), the user agent must handle the token as follows:

↪ **A character token that is U+0000 NULL**

[parse error](#). Ignore the token.

↪ **Any other character token**

Append the character token to the [pending table character tokens](#) list.

↪ **Anything else**

If any of the tokens in the [pending table character tokens](#) list are character tokens that are not [space characters](#), then this is a [parse error](#); reprocess the character tokens in the [pending table character tokens](#) list using the rules given in the "anything else" entry in the "[in table](#)" insertion mode.

Otherwise, [insert the characters](#) given by the [pending table character tokens](#) list.

Switch the [insertion mode](#) to the [original insertion mode](#) and reprocess the token.

§ 8.2.5.4.11. THE "IN CAPTION" INSERTION MODE

When the user agent is to apply the rules for the "[in caption](#)" [insertion mode](#), the user agent must handle the token as follows:

↪ **An end tag whose tag name is "caption"**

If the [stack of open elements](#) does not [have a caption element in table scope](#), this is a [parse error](#); ignore the token. ([fragment case](#))

Otherwise:

[Generate implied end tags](#).

Now, if the [current node](#) is not a [<caption>](#) element, then this is a [parse error](#).

Pop elements from this stack until a [<caption>](#) element has been popped from the stack.

[Clear the list of active formatting elements up to the last marker](#).

Switch the [insertion mode](#) to "[in table](#)".

↪ **A start tag whose tag name is one of: "caption", "col", "colgroup", "tbody", "td", "tfoot", "th", "thead", "tr"**

↪ **An end tag whose tag name is "table"**

If the [stack of open elements](#) does not [have a caption element in table scope](#), this is a [parse error](#); ignore the token. ([fragment case](#))

Otherwise:

[Generate implied end tags](#).

Now, if the [current node](#) is not a [<caption>](#) element, then this is a [parse error](#).

Pop elements from this stack until a [<caption>](#) element has been popped from the stack.

[Clear the list of active formatting elements up to the last marker](#).

Switch the [insertion mode](#) to "[in table](#)".

Reprocess the token.

↪ **An end tag whose tag name is one of: "body", "col", "colgroup", "html", "tbody", "td", "tfoot", "th", "thead", "tr"**

[parse error](#). Ignore the token.

↪ **Anything else**

Process the token [using the rules for the "in body" insertion mode](#).

§ 8.2.5.4.12. THE "IN COLUMN GROUP" INSERTION MODE

When the user agent is to apply the rules for the ["in column group" insertion mode](#), the user agent must handle the token as follows:

- ↪ **A character token that is one of U+0009 CHARACTER TABULATION, U+000A LINE FEED (LF), U+000C FORM FEED (FF), U+000D CARRIAGE RETURN (CR), or U+0020 SPACE**

[Insert the character](#).

- ↪ **A comment token**

[Insert a comment](#).

- ↪ **A DOCTYPE token**

[parse error](#). Ignore the token.

- ↪ **A start tag whose tag name is "html"**

Process the token [using the rules for the "in body" insertion mode](#).

- ↪ **A start tag whose tag name is "col"**

[Insert an HTML element](#) for the token. Immediately pop the [current node](#) off the [stack of open elements](#).

[Acknowledge the token's self-closing flag](#), if it is set.

- ↪ **An end tag whose tag name is "colgroup"**

If the [current node](#) is not a [<colgroup>](#) element, then this is a [parse error](#); ignore the token.

Otherwise, pop the [current node](#) from the [stack of open elements](#). Switch the [insertion mode](#) to ["in table"](#).

- ↪ **An end tag whose tag name is "col"**

[parse error](#). Ignore the token.

- ↪ **A start tag whose tag name is "template"**

- ↪ **An end tag whose tag name is "template"**

Process the token [using the rules for the "in head" insertion mode](#).

- ↪ **An end-of-file token**

Process the token [using the rules for the "in body" insertion mode](#).

- ↪ **Anything else**

If the [current node](#) is not a [<colgroup>](#) element, then this is a [parse error](#); ignore the token.

Otherwise, pop the [current node](#) from the [stack of open elements](#).

Switch the [insertion mode](#) to ["in table"](#).

Reprocess the token.

§ 8.2.5.4.13. THE "IN TABLE BODY" INSERTION MODE

When the user agent is to apply the rules for the ["in table body" insertion mode](#), the user agent must handle the token as follows:

- ↪ **A start tag whose tag name is "tr"**

[Clear the stack back to a table body context](#). (See below.)

[Insert an HTML element](#) for the token, then switch the [insertion mode](#) to ["in row"](#).

- ↪ **A start tag whose tag name is one of: "th", "td"**

[parse error](#).

[Clear the stack back to a table body context.](#) (See below.)

[Insert an HTML element](#) for a "tr" start tag token with no attributes, then switch the [insertion mode](#) to "in row".

Reprocess the current token.

↪ **An end tag whose tag name is one of: "tbody", "tfoot", "thead"**

If the [stack of open elements](#) does not [have an element in table scope](#) that is an [HTML element](#) with the same tag name as the token, this is a [parse error](#); ignore the token.

Otherwise:

[Clear the stack back to a table body context.](#) (See below.)

Pop the [current node](#) from the [stack of open elements](#). Switch the [insertion mode](#) to "in table".

↪ **A start tag whose tag name is one of: "caption", "col", "colgroup", "tbody", "tfoot", "thead"**

↪ **An end tag whose tag name is "table"**

If the [stack of open elements](#) does not [have a tbody, thead, or tfoot element in table scope](#), this is a [parse error](#); ignore the token.

Otherwise:

[Clear the stack back to a table body context.](#) (See below.)

Pop the [current node](#) from the [stack of open elements](#). Switch the [insertion mode](#) to "in table".

Reprocess the token.

↪ **An end tag whose tag name is one of: "body", "caption", "col", "colgroup", "html", "td", "th", "tr"**

[parse error](#). Ignore the token.

↪ **Anything else**

Process the token [using the rules for the "in table" insertion mode](#).

When the steps above require the user agent to **clear the stack back to a table body context**, it means that the user agent must, while the [current node](#) is not a [<tbody>](#), [<tfoot>](#), [<thead>](#), [<template>](#), or [<html>](#) element, pop elements from the [stack of open elements](#).

NOTE:

The [current node](#) being an [<html>](#) element after this process is a [fragment case](#).

§ 8.2.5.4.14. THE "IN ROW" INSERTION MODE

When the user agent is to apply the rules for the "in row" [insertion mode](#), the user agent must handle the token as follows:

↪ **A start tag whose tag name is one of: "th", "td"**

[Clear the stack back to a table row context.](#) (See below.)

[Insert an HTML element](#) for the token, then switch the [insertion mode](#) to "in cell".

Insert a [marker](#) at the end of the [list of active formatting elements](#).

↪ **An end tag whose tag name is "tr"**

If the [stack of open elements](#) does not [have a tr element in table scope](#), this is a [parse error](#); ignore the token.

Otherwise:

[Clear the stack back to a table row context.](#) (See below.)

Pop the [current node](#) (which will be a [<tr>](#) element) from the [stack of open elements](#). Switch the [insertion mode](#) to "in table body".

↪ **A start tag whose tag name is one of: "caption", "col", "colgroup", "tbody", "tfoot", "thead", "tr"**

↪ **An end tag whose tag name is "table"**

If the [stack of open elements](#) does not [have a tr element in table scope](#), this is a [parse error](#); ignore the token.

Otherwise:

[Clear the stack back to a table row context](#). (See below.)

Pop the [current node](#) (which will be a [tr](#) element) from the [stack of open elements](#). Switch the [insertion mode](#) to ["in table body"](#).

Reprocess the token.

↪ **An end tag whose tag name is one of: "tbody", "tfoot", "thead"**

If the [stack of open elements](#) does not [have an element in table scope](#) that is an [HTML element](#) with the same tag name as the token, this is a [parse error](#); ignore the token.

If the [stack of open elements](#) does not [have a tr element in table scope](#), ignore the token.

Otherwise:

[Clear the stack back to a table row context](#). (See below.)

Pop the [current node](#) (which will be a [tr](#) element) from the [stack of open elements](#). Switch the [insertion mode](#) to ["in table body"](#).

Reprocess the token.

↪ **An end tag whose tag name is one of: "body", "caption", "col", "colgroup", "html", "td", "th"**

[parse error](#). Ignore the token.

↪ **Anything else**

Process the token [using the rules for the "in table" insertion mode](#).

When the steps above require the user agent to **clear the stack back to a table row context**, it means that the user agent must, while the [current node](#) is not a [tr](#), [template](#), or [html](#) element, pop elements from the [stack of open elements](#).

NOTE:

The [current node](#) being an [html](#) element after this process is a [fragment case](#).

§ 8.2.5.4.15. THE "IN CELL" INSERTION MODE

When the user agent is to apply the rules for the ["in cell" insertion mode](#), the user agent must handle the token as follows:

↪ **An end tag whose tag name is one of: "td", "th"**

If the [stack of open elements](#) does not [have an element in table scope](#) that is an [HTML element](#) with the same tag name as that of the token, then this is a [parse error](#); ignore the token.

Otherwise:

[Generate implied end tags](#).

Now, if the [current node](#) is not an [HTML element](#) with the same tag name as the token, then this is a [parse error](#).

Pop elements from the [stack of open elements](#) stack until an [HTML element](#) with the same tag name as the token has been popped from the stack.

[Clear the list of active formatting elements up to the last marker](#).

Switch the [insertion mode](#) to ["in row"](#).

↪ **A start tag whose tag name is one of: "caption", "col", "colgroup", "tbody", "td", "tfoot", "th", "thead", "tr"**

If the [stack of open elements](#) does *not* [have a td or th element in table scope](#), then this is a [parse error](#); ignore the token. ([fragment case](#))

Otherwise, [close the cell](#) (see below) and reprocess the token.

- ↪ **An end tag whose tag name is one of: "body", "caption", "col", "colgroup", "html"**

[parse error](#). Ignore the token.

- ↪ **An end tag whose tag name is one of: "table", "tbody", "tfoot", "thead", "tr"**

If the [stack of open elements](#) does not [have an element in table scope](#) that is an [HTML element](#) with the same tag name as that of the token, then this is a [parse error](#); ignore the token.

Otherwise, [close the cell](#) (see below) and reprocess the token.

- ↪ **Anything else**

Process the token [using the rules for the "in body" insertion mode](#).

Where the steps above say to **close the cell**, they mean to run the following algorithm:

1. [Generate implied end tags](#).
2. If the [current node](#) is not now a [<td>](#) element or a [th](#) element, then this is a [parse error](#).
3. Pop elements from the [stack of open elements](#) stack until a [td](#) element or a [<th>](#) element has been popped from the stack.
4. [Clear the list of active formatting elements up to the last marker](#).
5. Switch the [insertion mode](#) to ["in row"](#).

NOTE:

The [stack of open elements](#) cannot have both a [<td>](#) and a [<th>](#) element [in table scope](#) at the same time, nor can it have neither when the [close the cell](#) algorithm is invoked.

§ 8.2.5.4.16. THE "IN SELECT" INSERTION MODE

When the user agent is to apply the rules for the ["in select" insertion mode](#), the user agent must handle the token as follows:

- ↪ **A character token that is U+0000 NULL**

[parse error](#). Ignore the token.

- ↪ **Any other character token**

[Insert the token's character](#).

- ↪ **A comment token**

[Insert a comment](#).

- ↪ **A DOCTYPE token**

[parse error](#). Ignore the token.

- ↪ **A start tag whose tag name is "html"**

Process the token [using the rules for the "in body" insertion mode](#).

- ↪ **A start tag whose tag name is "option"**

If the [current node](#) is an [<option>](#) element, pop that node from the [stack of open elements](#).

[Insert an HTML element](#) for the token.

- ↪ **A start tag whose tag name is "optgroup"**

If the [current node](#) is an [<option>](#) element, pop that node from the [stack of open elements](#).

If the [current node](#) is an [<optgroup>](#) element, pop that node from the [stack of open elements](#).

[Insert an HTML element](#) for the token.

- ↪ **An end tag whose tag name is "optgroup"**

First, if the [current node](#) is an `<option>` element, and the node immediately before it in the [stack of open elements](#) is an `optgroup` element, then pop the [current node](#) from the [stack of open elements](#).

If the [current node](#) is an `<optgroup>` element, then pop that node from the [stack of open elements](#). Otherwise, this is a [parse error](#); ignore the token.

↪ **An end tag whose tag name is "option"**

If the [current node](#) is an `<option>` element, then pop that node from the [stack of open elements](#). Otherwise, this is a [parse error](#); ignore the token.

↪ **An end tag whose tag name is "select"**

If the [stack of open elements](#) does not have a `<select>` element in select scope, this is a [parse error](#); ignore the token. ([fragment case](#))

Otherwise:

Pop elements from the [stack of open elements](#) until a `<select>` element has been popped from the stack.

[Reset the insertion mode appropriately.](#)

↪ **A start tag whose tag name is "select"**

[parse error.](#)

If the [stack of open elements](#) does not have a `<select>` element in select scope, ignore the token. ([fragment case](#))

Otherwise:

Pop elements from the [stack of open elements](#) until a `<select>` element has been popped from the stack.

[Reset the insertion mode appropriately.](#)

NOTE:

It just gets treated like an end tag.

↪ **A start tag whose tag name is one of: "input", "keygen", "textarea"**

[parse error.](#)

If the [stack of open elements](#) does not have a `<select>` element in select scope, ignore the token. ([fragment case](#))

Otherwise:

Pop elements from the [stack of open elements](#) until a `<select>` element has been popped from the stack.

[Reset the insertion mode appropriately.](#)

Reprocess the token.

↪ **A start tag whose tag name is one of: "script", "template"**

↪ **An end tag whose tag name is "template"**

Process the token [using the rules for the "in head" insertion mode](#).

↪ **An end-of-file token**

Process the token [using the rules for the "in body" insertion mode](#).

↪ **Anything else**

[parse error.](#) Ignore the token.

§ 8.2.5.4.17. THE "IN SELECT IN TABLE" INSERTION MODE

When the user agent is to apply the rules for the ["in select in table" insertion mode](#), the user agent must handle the token as follows:

↪ **A start tag whose tag name is one of: "caption", "table", "tbody", "tfoot", "thead", "tr", "td", "th"**

[parse error](#).

Pop elements from the [stack of open elements](#) until a `<select>` element has been popped from the stack.

[Reset the insertion mode appropriately](#).

Reprocess the token.

- ↪ **An end tag whose tag name is one of: "caption", "table", "tbody", "tfoot", "thead", "tr", "td", "th"**

[parse error](#).

If the [stack of open elements](#) does not [have an element in table scope](#) that is an [HTML element](#) with the same tag name as that of the token, then ignore the token.

Otherwise:

Pop elements from the [stack of open elements](#) until a `<select>` element has been popped from the stack.

[Reset the insertion mode appropriately](#).

Reprocess the token.

- ↪ **Anything else**

Process the token [using the rules for the "in select" insertion mode](#).

§ 8.2.5.4.18. THE "IN TEMPLATE" INSERTION MODE

When the user agent is to apply the rules for the ["in template" insertion mode](#), the user agent must handle the token as follows:

- ↪ **A character token**

- ↪ **A comment token**

- ↪ **A DOCTYPE token**

Process the token [using the rules for the "in body" insertion mode](#).

- ↪ **A start tag whose tag name is one of: "base", "basefont", "bgsound", "link", "meta", "noframes", "script", "style", "template", "title"**

- ↪ **An end tag whose tag name is "template"**

Process the token [using the rules for the "in head" insertion mode](#).

- ↪ **A start tag whose tag name is one of: "caption", "colgroup", "tbody", "tfoot", "thead"**

Pop the [current template insertion mode](#) off the [stack of template insertion modes](#).

Push ["in table"](#) onto the [stack of template insertion modes](#) so that it is the new [current template insertion mode](#).

Switch the [insertion mode](#) to ["in table"](#), and reprocess the token.

- ↪ **A start tag whose tag name is "col"**

Pop the [current template insertion mode](#) off the [stack of template insertion modes](#).

Push ["in column group"](#) onto the [stack of template insertion modes](#) so that it is the new [current template insertion mode](#).

Switch the [insertion mode](#) to ["in column group"](#), and reprocess the token.

- ↪ **A start tag whose tag name is "tr"**

Pop the [current template insertion mode](#) off the [stack of template insertion modes](#).

Push ["in table body"](#) onto the [stack of template insertion modes](#) so that it is the new [current template insertion mode](#).

Switch the [insertion mode](#) to ["in table body"](#), and reprocess the token.

- ↪ **A start tag whose tag name is one of: "td", "th"**

Pop the [current template insertion mode](#) off the [stack of template insertion modes](#).

Push "[in row](#)" onto the [stack of template insertion modes](#) so that it is the new [current template insertion mode](#).

Switch the [insertion mode](#) to "[in row](#)", and reprocess the token.

↪ **Any other start tag**

Pop the [current template insertion mode](#) off the [stack of template insertion modes](#).

Push "[in body](#)" onto the [stack of template insertion modes](#) so that it is the new [current template insertion mode](#).

Switch the [insertion mode](#) to "[in body](#)", and reprocess the token.

↪ **Any other end tag**

[parse error](#). Ignore the token.

↪ **An end-of-file token**

If there is no [<template>](#) element on the [stack of open elements](#), then [stop parsing](#). (fragment case)

Otherwise, this is a [parse error](#).

Pop elements from the [stack of open elements](#) until a [template](#) element has been popped from the stack.

[Clear the list of active formatting elements up to the last marker](#).

Pop the [current template insertion mode](#) off the [stack of template insertion modes](#).

[Reset the insertion mode appropriately](#).

Reprocess the token.

§ 8.2.5.4.19. THE "AFTER BODY" INSERTION MODE

When the user agent is to apply the rules for the "[after body](#)" [insertion mode](#), the user agent must handle the token as follows:

↪ **A character token that is one of U+0009 CHARACTER TABULATION, U+000A LINE FEED (LF), U+000C FORM FEED (FF), U+000D CARRIAGE RETURN (CR), or U+0020 SPACE**

Process the token [using the rules for the "in body" insertion mode](#).

↪ **A comment token**

[Insert a comment](#) as the last child of the first element in the [stack of open elements](#) (the [<html>](#) element).

↪ **A DOCTYPE token**

[parse error](#). Ignore the token.

↪ **A start tag whose tag name is "html"**

Process the token [using the rules for the "in body" insertion mode](#).

↪ **An end tag whose tag name is "html"**

If the parser was originally created as part of the [HTML fragment parsing algorithm](#), this is a [parse error](#); ignore the token. (fragment case)

Otherwise, switch the [insertion mode](#) to "[after after body](#)".

↪ **An end-of-file token**

[Stop parsing](#).

↪ **Anything else**

[parse error](#). Switch the [insertion mode](#) to "[in body](#)" and reprocess the token.

§ 8.2.5.4.20. THE "IN FRAMESET" INSERTION MODE

When the user agent is to apply the rules for the "[in frameset](#)" [insertion mode](#), the user agent must handle the token as follows:

- ↪ A character token that is one of U+0009 CHARACTER TABULATION, U+000A LINE FEED (LF), U+000C FORM FEED (FF), U+000D CARRIAGE RETURN (CR), or U+0020 SPACE
 - [Insert the character.](#)
 - ↪ A comment token
 - [Insert a comment.](#)
 - ↪ A DOCTYPE token
 - [parse error.](#) Ignore the token.
 - ↪ A start tag whose tag name is "html"
 - Process the token [using the rules for the "in body" insertion mode.](#)
 - ↪ A start tag whose tag name is "frameset"
 - [Insert an HTML element](#) for the token.
 - ↪ An end tag whose tag name is "frameset"
 - If the [current node](#) is the root [<html>](#) element, then this is a [parse error](#); ignore the token. ([fragment case](#))
 - Otherwise, pop the [current node](#) from the [stack of open elements](#).
 - If the parser was *not* originally created as part of the [HTML fragment parsing algorithm](#) ([fragment case](#)), and the [current node](#) is no longer a [<frameset>](#) element, then switch the [insertion mode](#) to "[after frameset](#)".
 - ↪ A start tag whose tag name is "frame"
 - [Insert an HTML element](#) for the token. Immediately pop the [current node](#) off the [stack of open elements](#).
 - [Acknowledge the token's self-closing flag](#), if it is set.
 - ↪ A start tag whose tag name is "noframes"
 - Process the token [using the rules for the "in head" insertion mode.](#)
 - ↪ An end-of-file token
 - If the [current node](#) is not the root [<html>](#) element, then this is a [parse error](#).
- NOTE:**
The [current node](#) can only be the root [<html>](#) element in the [fragment case](#).
- [Stop parsing.](#)
 - ↪ Anything else
 - [parse error.](#) Ignore the token.

§ 8.2.5.4.21. THE "AFTER FRAMESET" INSERTION MODE

When the user agent is to apply the rules for the "[after frameset](#)" [insertion mode](#), the user agent must handle the token as follows:

- ↪ A character token that is one of U+0009 CHARACTER TABULATION, U+000A LINE FEED (LF), U+000C FORM FEED (FF), U+000D CARRIAGE RETURN (CR), or U+0020 SPACE
 - [Insert the character.](#)
- ↪ A comment token
 - [Insert a comment.](#)
- ↪ A DOCTYPE token
 - [parse error.](#) Ignore the token.
- ↪ A start tag whose tag name is "html"

Process the token [using the rules for the "in body" insertion mode](#).

↪ **An end tag whose tag name is "html"**

Switch the [insertion mode](#) to ["after after frameset"](#).

↪ **A start tag whose tag name is "noframes"**

Process the token [using the rules for the "in head" insertion mode](#).

↪ **An end-of-file token**

[Stop parsing](#).

↪ **Anything else**

[parse error](#). Ignore the token.

§ 8.2.5.4.22. THE "AFTER AFTER BODY" INSERTION MODE

When the user agent is to apply the rules for the ["after after body" insertion mode](#), the user agent must handle the token as follows:

↪ **A comment token**

[Insert a comment](#) as the last child of the Document object.

↪ **A DOCTYPE token**

↪ **A character token that is one of U+0009 CHARACTER TABULATION, U+000A LINE FEED (LF), U+000C FORM FEED (FF), U+000D CARRIAGE RETURN (CR), or U+0020 SPACE**

↪ **A start tag whose tag name is "html"**

Process the token [using the rules for the "in body" insertion mode](#).

↪ **An end-of-file token**

[Stop parsing](#).

↪ **Anything else**

[parse error](#). Switch the [insertion mode](#) to ["in body"](#) and reprocess the token.

§ 8.2.5.4.23. THE "AFTER AFTER FRAMESET" INSERTION MODE

When the user agent is to apply the rules for the ["after after frameset" insertion mode](#), the user agent must handle the token as follows:

↪ **A comment token**

[Insert a comment](#) as the last child of the [Document](#) object.

↪ **A DOCTYPE token**

↪ **A character token that is one of U+0009 CHARACTER TABULATION, U+000A LINE FEED (LF), U+000C FORM FEED (FF), U+000D CARRIAGE RETURN (CR), or U+0020 SPACE**

↪ **A start tag whose tag name is "html"**

Process the token [using the rules for the "in body" insertion mode](#).

↪ **An end-of-file token**

[Stop parsing](#).

↪ **A start tag whose tag name is "noframes"**

Process the token [using the rules for the "in head" insertion mode](#).

↪ **Anything else**

[parse error](#). Ignore the token.

§ 8.2.5.5. The rules for parsing tokens in foreign content

When the user agent is to apply the rules for parsing tokens in foreign content, the user agent must handle the token as follows:

↪ **A character token that is U+0000 NULL**

[parse error](#). Insert a [U+FFFD REPLACEMENT CHARACTER](#) character.

↪ **A character token that is one of U+0009 CHARACTER TABULATION, U+000A LINE FEED (LF), U+000C FORM FEED (FF), U+000D CARRIAGE RETURN (CR), or U+0020 SPACE**

[Insert the token's character](#).

↪ **Any other character token**

[Insert the token's character](#).

Set the [frameset-ok flag](#) to "not ok".

↪ **A comment token**

[Insert a comment](#).

↪ **A DOCTYPE token**

[parse error](#). Ignore the token.

↪ **A start tag whose tag name is one of: "b", "big", "blockquote", "body", "br", "center", "code", "dd", "div", "dl", "dt", "em", "embed", "h1", "h2", "h3", "h4", "h5", "h6", "head", "hr", "i", "img", "li", "listing", "menu", "meta", "nobr", "ol", "p", "pre", "ruby", "s", "small", "span", "strong", "strike", "sub", "sup", "table", "tt", "u", "ul", "var"**

↪ **A start tag whose tag name is "font", if the token has any attributes named "color", "face", or "size"**

[parse error](#).

If the parser was originally created for the [HTML fragment parsing algorithm](#), then act as described in the "any other start tag" entry below. ([fragment case](#))

Otherwise:

Pop an element from the [stack of open elements](#), and then keep popping more elements from the [stack of open elements](#) until the [current node](#) is a [MathML text integration point](#), an [HTML integration point](#), or an element in the [HTML namespace](#).

Then, reprocess the token.

↪ **Any other start tag**

If the [adjusted current node](#) is an element in the [MathML namespace](#), [adjust MathML attributes](#) for the token. (This fixes the case of MathML attributes that are not all lowercase.)

If the [adjusted current node](#) is an element in the [SVG namespace](#), and the token's tag name is one of the ones in the first column of the following table, change the tag name to the name given in the corresponding cell in the second column. (This fixes the case of SVG elements that are not all lowercase.)

Tag name	Element name
altglyph	altGlyph
altglyphdef	altGlyphDef
altglyphitem	altGlyphItem
animatecolor	animateColor
animatemotion	animateMotion
animatetransform	animateTransform
clippath	clipPath
feblend	feBlend
fecolormatrix	feColorMatrix
fecomponenttransfer	feComponentTransfer
fecomposite	feComposite

Tag name	Element name
feconvolvematrix	feConvolveMatrix
fediffuselighting	feDiffuseLighting
fedisplacementmap	feDisplacementMap
fedistantlight	feDistantLight
fedropshadow	feDropShadow
feflood	feFlood
fefunca	feFuncA
fefuncb	feFuncB
fefuncg	feFuncG
fefuncr	feFuncR
fegaussianblur	feGaussianBlur
feimage	feImage
femerge	feMerge
femergenode	feMergeNode
femorphology	feMorphology
feoffset	feOffset
fepointlight	fePointLight
fespecularlighting	feSpecularLighting
fespotlight	feSpotLight
fetile	feTile
feturbulence	feTurbulence
foreignobject	foreignObject
glyphref	glyphRef
lineargradient	linearGradient
radialgradient	radialGradient
textpath	textPath

If the [adjusted current node](#) is an element in the [SVG namespace](#), [adjust SVG attributes](#) for the token. (This fixes the case of SVG attributes that are not all lowercase.)

[Adjust foreign attributes](#) for the token. (This fixes the use of namespaced attributes, in particular XLink in SVG.)

[Insert a foreign element](#) for the token, in the same namespace as the [adjusted current node](#).

If the token has its *self-closing flag* set, then run the appropriate steps from the following list:

↪ If the token's tag name is "script", and the new [current node](#) is in the [SVG namespace](#)
[Acknowledge the token's self-closing flag](#), and then act as described in the steps for a "script" end tag below.

↪ Otherwise
 Pop the [current node](#) off the [stack of open elements](#) and [acknowledge the token's self-closing flag](#).

↪ An end tag whose tag name is "script", if the [current node](#) is a [<script>](#) element in the [SVG namespace](#)

Pop the [current node](#) off the [stack of open elements](#).

Let the *old insertion point* have the same value as the current [insertion point](#). Let the [insertion point](#) be just before the [next input character](#).

Increment the parser's [script nesting level](#) by one. Set the [parser pause flag](#) to true.

[Process the script element](#) according to the SVG rules, if the user agent supports SVG. [SVG11]

NOTE:

Even if this causes [new characters to be inserted into the tokenizer](#), the parser will not be executed reentrantly, since the [parser pause flag](#) is true.

Decrement the parser's [script nesting level](#) by one. If the parser's [script nesting level](#) is zero, then set the [parser pause flag](#) to false.

Let the [insertion point](#) have the value of the [old insertion point](#). (In other words, restore the [insertion point](#) to its previous value. This value might be the "undefined" value.)

↪ Any other end tag

Run these steps:

1. Initialize [node](#) to be the [current node](#) (the bottommost node of the stack).
2. If [node](#)'s tag name, [converted to ASCII lowercase](#), is not the same as the tag name of the token, then this is a [parse error](#).
3. *Loop:* If [node](#) is the topmost element in the [stack of open elements](#), abort these steps. ([fragment case](#))
4. If [node](#)'s tag name, [converted to ASCII lowercase](#), is the same as the tag name of the token, pop elements from the [stack of open elements](#) until [node](#) has been popped from the stack, and then abort these steps.
5. Set [node](#) to the previous entry in the [stack of open elements](#).
6. If [node](#) is not an element in the [HTML namespace](#), return to the step labeled *loop*.
7. Otherwise, process the token according to the rules given in the section corresponding to the current [insertion mode](#) in HTML content.

§ 8.2.6. The end

Once the user agent **stops parsing** the document, the user agent must run the following steps:

1. Set the [current document readiness](#) to "interactive" and the [insertion point](#) to undefined.
2. Pop *all* the nodes off the [stack of open elements](#).
3. If the [list of scripts that will execute when the document has finished parsing](#) is not empty, run these substeps:
 1. [Spin the event loop](#) until the first [script](#) in the [list of scripts that will execute when the document has finished parsing](#) has its "[ready to be parser-executed](#)" flag set *and* the parser's Document [has no style sheet that is blocking scripts](#).
 2. [Execute](#) the first [script](#) in the [list of scripts that will execute when the document has finished parsing](#).
 3. Remove the first [script](#) element from the [list of scripts that will execute when the document has finished parsing](#) (i.e., shift out the first entry in the list).
 4. If the [list of scripts that will execute when the document has finished parsing](#) is still not empty, repeat these substeps again from substep 1.
4. [Queue a task](#) to [fire a simple event](#) that bubbles named DOMContentLoaded at the [Document](#).
5. [Spin the event loop](#) until the [set of scripts that will execute as soon as possible](#) and the [list of scripts that will execute in order as soon as possible](#) are empty.
6. [Spin the event loop](#) until there is nothing that **delays the load event** in the [Document](#).
7. [Queue a task](#) to run the following substeps:
 1. Set the [current document readiness](#) to "complete".
 2. *Load event:* If the Document is in a [browsing context](#), [fire a simple event](#) named load at the Document's Window object, with *target override* set to the Document object.

8. If the Document is in a [browsing context](#), then [queue a task](#) to run the following substeps:
 1. If the Document's [page showing](#) flag is true, then abort this task (i.e., don't fire the event below).
 2. Set the Document's [page showing](#) flag to true.
 3. **Fire** a [trusted](#) event with the name `pageshow` at the Window object of the [Document](#), with *target override* set to the Document object, using the `PageTransitionEvent` interface, with the `persisted` attribute initialized to false. This event must not bubble, must not be cancelable, and has no default action.
9. If the [Document](#) has any [pending application cache download process tasks](#), then [queue](#) each such [task](#) in the order they were added to the list of [pending application cache download process tasks](#), and then empty the list of [pending application cache download process tasks](#). The [task source](#) for these [tasks](#) is the [networking task source](#).
10. If the Document's [print when loaded](#) flag is set, then run the [printing steps](#).
11. The Document is now **ready for post-load tasks**.
12. [Queue a task](#) to mark the Document as **completely loaded**.

When the user agent is to **abort a parser**, it must run the following steps:

1. Throw away any pending content in the [input stream](#), and discard any future content that would have been added to it.
2. Set the [current document readiness](#) to "interactive".
3. Pop *all* the nodes off the [stack of open elements](#).
4. Set the [current document readiness](#) to "complete".

Except where otherwise specified, the [task source](#) for the [tasks](#) mentioned in this section is the [DOM manipulation task source](#).

§ 8.2.7. Coercing an HTML DOM into an infoset

When an application uses an [HTML parser](#) in conjunction with an XML pipeline, it is possible that the constructed DOM is not compatible with the XML tool chain in certain subtle ways. For example, an XML toolchain might not be able to represent attributes with the name `xmlns`, since they conflict with the Namespaces in XML syntax. There is also some data that the [HTML parser](#) generates that isn't included in the DOM itself. This section specifies some rules for handling these issues.

If the XML API being used doesn't support DOCTYPEs, the tool may drop DOCTYPEs altogether.

If the XML API doesn't support attributes in no namespace that are named "xmlns", attributes whose names start with "xmlns:", or attributes in the [XMLNS namespace](#), then the tool may drop such attributes.

The tool may annotate the output with any namespace declarations required for proper operation.

If the XML API being used restricts the allowable characters in the local names of elements and attributes, then the tool may map all element and attribute local names that the API wouldn't support to a set of names that *are* allowed, by replacing any character that isn't supported with the uppercase letter U and the six digits of the character's Unicode code point when expressed in hexadecimal, using digits 0-9 and capital letters A-F as the symbols, in increasing numeric order.

EXAMPLE 650

For example, the element name `foo<bar`, which can be output by the [HTML parser](#), though it is neither a legal HTML element name nor a well-formed XML element name, would be converted into `fooU00003Cbar`, which *is* a well-formed XML element name (though it's still not legal in HTML by any means).

EXAMPLE 651

As another example, consider the attribute `xlink:href`. Used on a MathML element, it becomes, after being [adjusted](#), an attribute with a prefix "xlink" and a local name "href". However, used on an HTML element, it becomes an attribute with no prefix and the local name "xlink:href", which is not a valid NCName, and thus might not be accepted by an XML API. It could thus get converted, becoming "xlinkU00003Ahref".

NOTE:

The resulting names from this conversion conveniently can't clash with any attribute generated by the [HTML parser](#), since those are all either lowercase or those listed in the [adjust foreign attributes](#) algorithm's table.

If the XML API restricts comments from having two consecutive U+002D HYPHEN-MINUS characters (--), the tool may insert a single U+0020 SPACE character between any such offending characters.

If the XML API restricts comments from ending in a U+002D HYPHEN-MINUS character (-), the tool may insert a single U+0020 SPACE character at the end of such comments.

If the XML API restricts allowed characters in character data, attribute values, or comments, the tool may replace any U+000C FORM FEED (FF) character with a U+0020 SPACE character, and any other literal non-XML character with a U+FFFD REPLACEMENT CHARACTER.

If the tool has no way to convey out-of-band information, then the tool may drop the following information:

- Whether the document is set to *no-quirks mode*, *limited-quirks mode*, or *quirks mode*
- The association between form controls and forms that aren't their nearest form element ancestor (use of the [form element pointer](#) in the parser)
- The [template contents](#) of any `<template>` elements.

NOTE:

The mutations allowed by this section apply after the [HTML parser](#)'s rules have been applied. For example, a `<a: :>` start tag will be closed by a `</a: :>` end tag, and never by a `</aU00003AU00003A>` end tag, even if the user agent is using the rules above to then generate an actual element in the DOM with the name `aU00003AU00003A` for that start tag.

§ 8.2.8. An introduction to error handling and strange cases in the parser

This section is non-normative.

This section examines some erroneous markup and discusses how the [HTML parser](#) handles these cases.

§ 8.2.8.1. Misnested tags: `<b><i></b></i>`

This section is non-normative.

The most-often discussed example of erroneous markup is as follows:

```
<p>1<b>2<i>3</b>4</i>5</p>
```

The parsing of this markup is straightforward up to the "3". At this point, the DOM looks like this:

```

L
├─<html>
├─┬─<head>
├─┬─<body>
├─┬─<p>
├─┬─┬─#text: 1
├─┬─┬─<b>
├─┬─┬─┬─#text: 2
├─┬─┬─┬─<i>
├─┬─┬─┬─┬─#text: 3

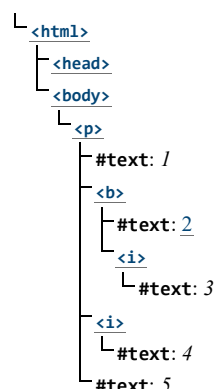
```

Here, the [stack of open elements](#) has five elements on it: `<html>`, `<body>`, `<p>`, `<b>`, and `<i>`. The [list of active formatting elements](#) just has two: `<b>` and `<i>`. The [insertion mode](#) is "in body".

Upon receiving the end tag token with the tag name "b", the [adoption agency algorithm](#) is invoked. This is a simple case, in that the *formatting element* is the `<b>` element, and there is no *furthest block*. Thus, the [stack of open elements](#) ends up

with just three elements: `<html>`, `<body>`, and `<p>`, while the [list of active formatting elements](#) has just one: `<i>`. The DOM tree is unmodified at this point.

The next token is a character ("4"), triggers the [reconstruction of the active formatting elements](#), in this case just the `<i>` element. A new `<i>` element is thus created for the "4" Text node. After the end tag token for the "i" is also received, and the "5" Text node is inserted, the DOM looks as follows:



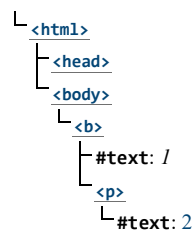
8.2.8.2. Misnested tags: `<b><p></b></p>`

This section is non-normative.

A case similar to the previous one is the following:

```
<b>1<p>2</b>3</p>
```

Up to the "2" the parsing here is straightforward:



The interesting part is when the end tag token with the tag name "b" is parsed.

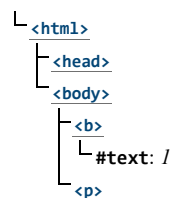
Before that token is seen, the [stack of open elements](#) has four elements on it: `<html>`, `<body>`, `<b>`, and `<p>`. The [list of active formatting elements](#) just has the one: `<b>`. The [insertion mode](#) is "in body".

Upon receiving the end tag token with the tag name "b", the "[adoption agency algorithm](#)" is invoked, as in the previous example. However, in this case, there is a *furthest block*, namely the `<p>` element. Thus, this time the adoption agency algorithm isn't skipped over.

The [common ancestor](#) is the `<body>` element. A conceptual "bookmark" marks the position of the b in the [list of active formatting elements](#), but since that list has only one element in it, the bookmark won't have much effect.

As the algorithm progresses, *node* ends up set to the formatting element (b), and *last node* ends up set to the *furthest block* (p).

The *last node* gets appended (moved) to the *common ancestor*, so that the DOM looks like:



```

  L #text: 2

```

A new `<b>` element is created, and the children of the `<p>` element are moved to it:

```

  L <html>
    L <head>
    L <body>
      L <b>
        L #text: 1
      L <p>
  L <b>
    L #text: 2

```

Finally, the new `<b>` element is appended to the `<p>` element, so that the DOM looks like:

```

  L <html>
    L <head>
    L <body>
      L <b>
        L #text: 1
      L <p>
        L <b>
          L #text: 2

```

The `<b>` element is removed from the [list of active formatting elements](#) and the [stack of open elements](#), so that when the "3" is parsed, it is appended to the `<p>` element:

```

  L <html>
    L <head>
    L <body>
      L <b>
        L #text: 1
      L <p>
        L <b>
          L #text: 2
        L #text: 3

```

§ 8.2.8.3. Unexpected markup in tables

This section is non-normative.

Error handling in tables is, for historical reasons, especially strange. For example, consider the following markup:

```
<table><b><tr><td>aaa</td></tr>bbb</table>ccc
```

The highlighted `<b>` element start tag is not allowed directly inside a table like that, and the parser handles this case by placing the element *before* the table. (This is called *foster parenting*.) This can be seen by examining the DOM tree as it stands just after the `<table>` element's start tag has been seen:

```

  L <html>
    L <head>
    L <body>
      L <table>

```

...and then immediately after the `<b>` element start tag has been seen:

```

  L <html>
    L <head>
    L <body>
      L <b>

```


[<table>](#)

At this point, the [stack of open elements](#) has on it the elements [<html>](#), [<body>](#), [<table>](#), and [b](#) (in that order, despite the resulting DOM tree); the [list of active formatting elements](#) just has the [](#) element in it; and the [insertion mode](#) is "in table".

The [tr](#) start tag causes the [](#) element to be popped off the stack and a [tbody](#) start tag to be implied; the [tbody](#) and [<tr>](#) elements are then handled in a rather straight-forward manner, taking the parser through the "in table body" and "in row" insertion modes, after which the DOM looks as follows:

```

L <html>
  |
  |_ <head>
  |
  |_ <body>
    |
    |_ <b>
    |
    |_ <table>
      |
      |_ <tbody>
        |
        |_ <tr>

```

Here, the [stack of open elements](#) has on it the elements [<html>](#), [<body>](#), [<table>](#), [<tbody>](#), and [tr](#); the [list of active formatting elements](#) still has the [](#) element in it; and the [insertion mode](#) is "in row".

The [<td>](#) element start tag token, after putting a [<td>](#) element on the tree, puts a [marker](#) on the [list of active formatting elements](#) (it also switches to the "in cell" insertion mode).

```

L <html>
  |
  |_ <head>
  |
  |_ <body>
    |
    |_ <b>
    |
    |_ <table>
      |
      |_ <tbody>
        |
        |_ <tr>
          |
          |_ <td>

```

The [marker](#) means that when the "aaa" character tokens are seen, no [](#) element is created to hold the resulting Text node:

```

L <html>
  |
  |_ <head>
  |
  |_ <body>
    |
    |_ <b>
    |
    |_ <table>
      |
      |_ <tbody>
        |
        |_ <tr>
          |
          |_ <td>
            |
            |_ #text: aaa

```

The end tags are handled in a straight-forward manner; after handling them, the [stack of open elements](#) has on it the elements [<html>](#), [<body>](#), [<table>](#), and [tbody](#); the [list of active formatting elements](#) still has the [](#) element in it (the [marker](#) having been removed by the "td" end tag token); and the [insertion mode](#) is "in table body".

Thus it is that the "bbb" character tokens are found. These trigger the "in table text" insertion mode to be used (with the [original insertion mode](#) set to "in table body"). The character tokens are collected, and when the next token (the [<table>](#) element end tag) is seen, they are processed as a group. Since they are not all spaces, they are handled as per the "anything else" rules in the "in table" insertion mode, which defer to the "in body" insertion mode but with [foster parenting](#).

When the [active formatting elements are reconstructed](#), a [](#) element is created and [foster parented](#), and then the "bbb" Text node is appended to it:

```

L <html>
  |
  |_ <head>
  |
  |_ <body>
    |
    |_ <b>
    |
    |_ <b>
      |
      |_ #text: bbb

```

```

-
  <table>
  |
  |_ <tbody>
  |   |
  |   |_ <tr>
  |       |
  |       |_ <td>
  |           |
  |           |_ #text: aaa

```

The [stack of open elements](#) has on it the elements [<html>](#), [<body>](#), [<table>](#), [<tbody>](#), and the new **b** (again, note that this doesn't match the resulting tree!); the [list of active formatting elements](#) has the new [](#) element in it; and the [insertion mode](#) is still "in table body".

Had the character tokens been only [space characters](#) instead of "bbb", then those [space characters](#) would just be appended to the [<tbody>](#) element.

Finally, the table is closed by a "table" end tag. This pops all the nodes from the [stack of open elements](#) up to and including the [<table>](#) element, but it doesn't affect the [list of active formatting elements](#), so the "ccc" character tokens after the table result in yet another [](#) element being created, this time after the table:

```

L
|_ <html>
|   |
|   |_ <head>
|       |
|       |_ <body>
|           |
|           |_ <b>
|               |
|               |_ <b>
|                   |
|                   |_ #text: bbb
|                       |
|                       |_ <table>
|                           |
|                           |_ <tbody>
|                               |
|                               |_ <tr>
|                                   |
|                                   |_ <td>
|                                       |
|                                       |_ #text: aaa
|                                           |
|                                           |_ <b>
|                                               |
|                                               |_ #text: ccc

```

§ 8.2.8.4. Scripts that modify the page as it is being parsed

This section is non-normative.

Consider the following markup, which for this example we will assume is the document with [URL](#) <https://example.com> /inner, being rendered as the content of an `iframe` in another document with the [URL](#) <https://example.com/outer>:

```

<div id=a>
<script>
var div = document.getElementById("a");
parent.document.body.appendChild(div);
</script>
<script>
alert(document.URL);
</script>
</div>
<script>
alert(document.URL);
</script>

```

Up to the first "script" end tag, before the script is parsed, the result is relatively straightforward:

```

L
|_ <html>
|   |
|   |_ <head>
|       |
|       |_ <body>
|           |
|           |_ <div id="a">
|               |
|               |_ #text:
|                   |
|                   |_ <script>
|                       |
|                       |_ #text: var div = document.getElementById("a"); ↵ parent.document.body.appendChild(div);

```

After the script is parsed, though, the `<div>` element and its child `<script>` element are gone:

```

L <html>
  |
  | <head>
  |
  | <body>

```

They are, at this point, in the Document of the aforementioned outer [browsing context](#). However, the [stack of open elements](#) still contains the `<div>` element.

Thus, when the second `<script>` element is parsed, it is inserted *into the outer Document object*.

Those parsed into different Documents than the one the parser was created for do not execute, so the first alert does not show.

Once the `<div>` element's end tag is parsed, the `<div>` element is popped off the stack, and so the next `<script>` element is in the inner Document:

```

L <html>
  |
  | <head>
  |
  | <body>
    |
    | <script>
      |
      | #text: alert(document.URL);

```

This script does execute, resulting in an alert that says "https://example.com/inner".

8.2.8.5. The execution of scripts that are moving across multiple documents

This section is non-normative.

Elaborating on the example in the previous section, consider the case where the second `<script>` element is an external script (i.e., one with a `src` attribute). Since the element was not in the parser's Document when it was created, that external script is not even downloaded.

In a case where a `<script>` element with a `src` attribute is parsed normally into its parser's [Document](#), but while the external script is being downloaded, the element is moved to another document, the script continues to download, but does not execute.

NOTE:

In general, moving `<script>` elements between Documents is considered a bad practice.

8.2.8.6. Unclosed formatting elements

This section is non-normative.

The following markup shows how nested formatting elements (such as `b`) get collected and continue to be applied even as the elements they are contained in are closed, but that excessive duplicates are thrown away.

```

<!DOCTYPE html>
<p><b class=x><b class=x><b><b class=x><b class=x><b>X
<p>X
<p><b><b class=x><b>X
<p></b></b></b></b></b></b></b></b>X

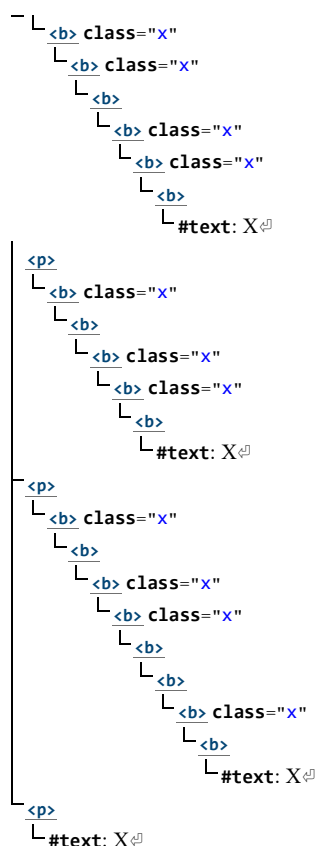
```

The resulting DOM tree is as follows:

```

L DOCTYPE: html
  |
  | <html>
  |
  | <head>
  |
  | <body>
    |
    | <p>

```



Note how the second `<p>` element in the markup has no explicit `b` elements, but in the resulting DOM, up to three of each kind of formatting element (in this case three `<b>` elements with the `class` attribute, and two unadorned `<b>` elements) get reconstructed before the element's "X".

Also note how this means that in the final paragraph only six `b` end tags are needed to completely clear the [list of active formatting elements](#), even though nine `b` start tags have been seen up to this point.

§ 8.3. Serializing HTML fragments

The following steps form the **HTML fragment serialization algorithm**. The algorithm takes as input a DOM [Element](#), [Document](#), or [DocumentFragment](#) referred to as *the node*, and either returns a string.

NOTE:

This algorithm serializes the children of the node being serialized, not the node itself.

1. Let `s` be a string, and initialize it to the empty string.
2. If *the node* is a `<template>` element, then let *the node* instead be the `<template>` element's [template contents](#) (a `DocumentFragment` node).
3. For each child node of *the node*, in [tree order](#), run the following steps:
 1. Let *current node* be the child node being processed.
 2. Append the appropriate string from the following list to `s`:

↪ If *current node* is an [Element](#)

If *current node* is an element in the [HTML namespace](#), the [MathML namespace](#), or the [SVG namespace](#), then let *tagname* be *current node*'s local name. Otherwise, let *tagname* be *current node*'s qualified name.

Append a U+003C LESS-THAN SIGN character (`<`), followed by *tagname*.

NOTE:

For [html elements](#) created by the [HTML parser](#) or `Document.createElement()`, *tagname* will be lowercase.

For each attribute that the element has, append a U+0020 SPACE character, the [attribute's serialized name as described below](#), a U+003D EQUALS SIGN character (=), a U+0022 QUOTATION MARK character ("), the attribute's value, [escaped as described below](#) in *attribute mode*, and a second U+0022 QUOTATION MARK character (").

An **attribute's serialized name** for the purposes of the previous paragraph must be determined as follows:

↪ **If the attribute has no namespace**

The attribute's serialized name is the attribute's local name.

NOTE:

For attributes on [html elements](#) set by the [HTML parser](#) or by `Element.setAttribute()`, the local name will be lowercase.

↪ **If the attribute is in the [XML namespace](#)**

The attribute's serialized name is the string "xml:" followed by the attribute's local name.

↪ **If the attribute is in the [XMLNS namespace](#) and the attribute's local name is `xmlns`**

The attribute's serialized name is the string "xmlns".

↪ **If the attribute is in the [XMLNS namespace](#) and the attribute's local name is not `xmlns`**

The attribute's serialized name is the string "xmlns:" followed by the attribute's local name.

↪ **If the attribute is in the [XLink namespace](#)**

The attribute's serialized name is the string "xlink:" followed by the attribute's local name.

↪ **If the attribute is in some other namespace**

The attribute's serialized name is the attribute's qualified name.

While the exact order of attributes is user agent-defined, and may depend on factors such as the order that the attributes were given in the original markup, the sort order must be stable, such that consecutive invocations of this algorithm serialize an element's attributes in the same order.

Append a U+003E GREATER-THAN SIGN character (>).

If *current node* is an `<area>`, `<base>`, `<basefont>`, `<bgsound>`, `<br>`, `<col>`, `<embed>`, `<frame>`, `<hr>`, `<img>`, `<input>`, `<keygen>`, `<link>`, `<menuitem>`, `<meta>`, `<param>`, `<source>`, `<track>` or `<wbr>` element, then continue on to the next child node at this point.

If *current node* is a `<pre>`, `<textarea>`, or `<listing>` element, and the first child node of the element, if any, is a Text node whose character data has as its first character a U+000A LINE FEED (LF) character, then append a U+000A LINE FEED (LF) character.

Append the value of running the [HTML fragment serialization algorithm](#) on the *current node* element (thus recursing into this algorithm for that element), followed by a U+003C LESS-THAN SIGN character (<), a U+002F SOLIDUS character (/), *tagname* again, and finally a U+003E GREATER-THAN SIGN character (>).

↪ **If *current node* is a Text node**

If the parent of *current node* is a `<style>`, `<script>`, `<xmp>`, `<iframe>`, `<noembed>`, `<noframes>`, or `<plaintext>` element, or if the parent of *current node* is a `<noscript>` element and [scripting is enabled](#) for the node, then append the value of *current node*'s data IDL attribute literally.

Otherwise, append the value of *current node*'s data IDL attribute, [escaped as described below](#).

↪ **If *current node* is a Comment**

Append the literal string "<!--" (U+003C LESS-THAN SIGN, U+0021 EXCLAMATION MARK, U+002D HYPHEN-MINUS, U+002D HYPHEN-MINUS), followed by the value of *current node*'s

data IDL attribute, followed by the literal string "-->" (U+002D HYPHEN-MINUS, U+002D HYPHEN-MINUS, U+003E GREATER-THAN SIGN).

↪ If *current node* is a **ProcessingInstruction**

Append the literal string "<?" (U+003C LESS-THAN SIGN, U+003F QUESTION MARK), followed by the value of *current node*'s **target** IDL attribute, followed by a single U+0020 SPACE character, followed by the value of *current node*'s **data** IDL attribute, followed by a single U+003E GREATER-THAN SIGN character (>).

↪ If *current node* is a **DocumentType**

Append the literal string "<!DOCTYPE" (U+003C LESS-THAN SIGN, U+0021 EXCLAMATION MARK, U+0044 LATIN CAPITAL LETTER D, U+004F LATIN CAPITAL LETTER O, U+0043 LATIN CAPITAL LETTER C, U+0054 LATIN CAPITAL LETTER T, U+0059 LATIN CAPITAL LETTER Y, U+0050 LATIN CAPITAL LETTER P, U+0045 LATIN CAPITAL LETTER E), followed by a space (U+0020 SPACE), followed by the value of *current node*'s **name** IDL attribute, followed by the literal string ">" (U+003E GREATER-THAN SIGN).

4. The result of the algorithm is the string *s*.

⚠Warning! It is possible that the output of this algorithm, if parsed with an [HTML parser](#), will not return the original tree structure.

EXAMPLE 652

For instance, if a `<textarea>` element to which a **Comment** node has been appended is serialized and the output is then reparsed, the comment will end up being displayed in the text field. Similarly, if, as a result of DOM manipulation, an element contains a comment that contains the literal string "-->", then when the result of serializing the element is parsed, the comment will be truncated at that point and the rest of the comment will be interpreted as markup. More examples would be making a `<script>` element contain a **Text** node with the text string "</script>", or having a `<p>` element that contains a `<ul>` element (as the `ul` element's **start tag** would imply the end tag for the `p`).

This can enable cross-site scripting attacks. An example of this would be a page that lets the user enter some font family names that are then inserted into a CSS `style` block via the DOM and which then uses the **innerHTML** IDL attribute to get the HTML serialization of that `<style>` element: if the user enters "`</style><script>attack</script>`" as a font family name, **innerHTML** will return markup that, if parsed in a different context, would contain a `<script>` node, even though no `<script>` node existed in the original DOM.

Escaping a string (for the purposes of the algorithm above) consists of running the following steps:

1. Replace any occurrence of the "&" character by the string "&"
2. Replace any occurrences of the U+00A0 NO-BREAK SPACE character by the string " "
3. If the algorithm was invoked in the *attribute mode*, replace any occurrences of the "" character by the string """
4. If the algorithm was *not* invoked in the *attribute mode*, replace any occurrences of the "<" character by the string "<", and any occurrences of the ">" character by the string ">"

§ 8.4. Parsing HTML fragments

The following steps form the **HTML fragment parsing algorithm**. The algorithm takes as input an **Element** node, referred to as the **context** element, which gives the context for the parser, as well as *input*, a string to parse, and returns a list of zero or more nodes.

NOTE:

Parts marked **fragment case** in algorithms in the parser section are parts that only occur if the parser was created for the purposes of this algorithm. The algorithms have been annotated with such markings for informational purposes only; such markings have no normative weight. If it is possible for a condition described as a [fragment case](#) to occur even when the parser wasn't created for the purposes of handling this algorithm, then that is an error in the specification.

1. Create a new Document node, and mark it as being an [HTML document](#).
2. If the [node document](#) of the `context` element is in [quirks mode](#), then let the Document be in [quirks mode](#). Otherwise, the [node document](#) of the `context` element is in [limited-quirks mode](#), then let the Document be in [limited-quirks mode](#). Otherwise, leave the Document in [no-quirks mode](#).
3. Create a new [HTML parser](#), and associate it with the just created Document node.
4. Set the state of the [HTML parser](#)'s [tokenization](#) stage as follows:

↪ If it is a [title](#) or [<textarea>](#) element

Switch the tokenizer to the [§8.2.4.3 RCDATA state](#).

↪ If it is a [<style>](#), [<xmp>](#), [<iframe>](#), [<noembed>](#), or [<noframes>](#) element

Switch the tokenizer to the [§8.2.4.5 RAWTEXT state](#).

↪ If it is a [<script>](#) element

Switch the tokenizer to the [§8.2.4.6 Script data state](#).

↪ If it is a [<noscript>](#) element

If the [scripting flag](#) is enabled, switch the tokenizer to the [§8.2.4.5 RAWTEXT state](#). Otherwise, leave the tokenizer in the [§8.2.4.1 Data state](#).

↪ If it is a [<plaintext>](#) element

Switch the tokenizer to the [§8.2.4.7 PLAINTEXT state](#).

↪ Otherwise

Leave the tokenizer in the [§8.2.4.1 Data state](#).

NOTE:

For performance reasons, an implementation that does not report errors and that uses the actual state machine described in this specification directly could use the PLAINTEXT state instead of the RAWTEXT and script data states where those are mentioned in the list above. Except for rules regarding parse errors, they are equivalent, since there is no [appropriate end tag token](#) in the fragment case, yet they involve far fewer state transitions.

5. Let `root` be a new [<html>](#) element with no attributes.
6. Append the element `root` to the Document node created above.
7. Set up the parser's [stack of open elements](#) so that it contains just the single element `root`.
8. If the `context` element is a [<template>](#) element, push "[in template](#)" onto the [stack of template insertion modes](#) so that it is the new [current template insertion mode](#).
9. Create a start tag token whose name is the local name of `context` and whose attributes are the attributes of `context`.

Let this start tag token be the start tag token of the `context` node, e.g., for the purposes of determining if it is an [HTML integration point](#).
10. [Reset the parser's insertion mode appropriately](#).

NOTE:

The parser will reference the `context` element as part of that algorithm.

11. Set the parser's [form element pointer](#) to the nearest node to the `context` element that is a [form element](#) (going straight up the ancestor chain, and including the element itself, if it is a [<form>](#) element), if any. (If there is no such [<form>](#) ele-

ment, the [form element pointer](#) keeps its initial value, null.)

12. Place the [input](#) into the [input stream](#) for the [HTML parser](#) just created. The encoding [confidence](#) is *irrelevant*.
13. Start the parser and let it run until it has consumed all the characters just inserted into the input stream.
14. Return the child nodes of [root](#), in [tree order](#).

§ 8.5. Named character references

This table lists the character reference names that are supported by HTML, and the code points to which they refer. It is referenced by the previous sections.

Name	Character(s)	Glyph
Aacute;	U+000C1	Á
Aacute	U+000C1	Á
acute;	U+000E1	á
acute	U+000E1	á
Abreve;	U+00102	Ă
abreve;	U+00103	ă
ac;	U+0223E	˜
acd;	U+0223F	˘
acE;	U+0223E U+00333	˜
Acirc;	U+000C2	Â
Acirc	U+000C2	Â
acirc;	U+000E2	â
acirc	U+000E2	â
acute;	U+000B4	´
acute	U+000B4	´
Acy;	U+00410	А
acy;	U+00430	а
AElig;	U+000C6	Æ
AElig	U+000C6	Æ
aelig;	U+000E6	æ
aelig	U+000E6	æ
af;	U+02061	
Afr;	U+1D504	𝐀
afr;	U+1D51E	𝐚
Agrave;	U+000C0	À
Agrave	U+000C0	À
agrave;	U+000E0	à
agrave	U+000E0	à
alefsym;	U+02135	ℵ
aleph;	U+02135	ℵ
Alpha;	U+00391	Α
alpha;	U+003B1	α
Amacr;	U+00100	Ā
amacr;	U+00101	ā
amalg;	U+02A3F	ℳ
AMP;	U+00026	&
AMP	U+00026	&
amp;	U+00026	&
amp	U+00026	&
And;	U+02A53	ℳ
and;	U+02227	∧
andand;	U+02A55	ℳ
andd;	U+02A5C	ℳ
andslope;	U+02A58	ℳ
andv;	U+02A5A	ℳ
ang;	U+02220	∠
ange;	U+029A4	⤵
angle;	U+02220	∠
angmsd;	U+02221	∠
angmsdaa;	U+029A8	⤵
angmsdab;	U+029A9	⤵
angmsdac;	U+029AA	⤵
angmsdad;	U+029AB	⤵
angmsdae;	U+029AC	⤵
angmsdaf;	U+029AD	⤵
angmsdag;	U+029AE	⤵
angmsdah;	U+029AF	⤵
angrt;	U+0221F	└
angrtvrb;	U+022BE	⋈
angrtvbd;	U+0299D	⋈
angsph;	U+02222	⋈
angst;	U+000C5	Å
angzarr;	U+0237C	Ɽ
Aogon;	U+00104	Ą
aogon;	U+00105	ą
Aopf;	U+1D538	𝐀
aopf;	U+1D552	𝐚
ap;	U+02248	≈
apacir;	U+02A6F	⌢
apE;	U+02A70	⌢
ape;	U+0224A	≐
apid;	U+0224B	≐
apos;	U+00027	'

This data is also available [as a JSON file](#).

The glyphs displayed above are non-normative. Refer to the Unicode specifications for formal definitions of the characters listed above.

NOTE:

The character reference names originate from the *XML Entity Definitions for Characters* specification, though only the above is considered normative. [\[\[XML-ENTITY-NAMES\]\]](#)

§ 9. The XHTML syntax

NOTE:

This section only describes the rules for XML resources. Rules for [text/html](#) resources are discussed in the section above entitled "[The HTML syntax](#)".

§ 9.1. Writing XHTML documents

The syntax for using HTML with XML, whether in XHTML documents or embedded in other XML documents, is defined in the XML and Namespaces in XML specifications. [\[XML\]](#) [\[XML-NAMES\]](#)

This specification does not define any syntax-level requirements beyond those defined for XML proper.

XML documents may contain a DOCTYPE if desired, but this is not required to conform to this specification. This specification does not define a public or system identifier, nor provide a formal DTD.

NOTE:

According to the XML specification, XML processors are not guaranteed to process the external DTD subset referenced in the DOCTYPE. This means, for example, that using [entity references](#) for characters in XHTML documents is unsafe if they are defined in an external file (except for `<`, `>`, `&`, `"` and `'`).

§ 9.2. Parsing XHTML documents

This section describes the relationship between XML and the DOM, with a particular emphasis on how this interacts with HTML.

An **XML parser**, for the purposes of this specification, is a construct that follows the rules given in the XML specification to map a string of bytes or characters into a Document object.

NOTE:

At the time of writing, no such rules actually exist.

An [XML parser](#) is either associated with a Document object when it is created, or creates one implicitly.

This Document must then be populated with DOM nodes that represent the tree structure of the input passed to the parser, as defined by the XML specification, the Namespaces in XML specification, and the DOM specification. DOM mutation events must not fire for the operations that the [XML parser](#) performs on the Document's tree, but the user agent must act as if elements and attributes were individually appended and set respectively so as to trigger rules in this specification regarding what happens when an element is inserted into a document or has its attributes set, and the DOM specification's requirements regarding mutation observers mean that mutation observers *are* fired (unlike mutation events). [\[XML\]](#) [\[XML-NAMES\]](#) [\[DOM\]](#) [\[UIEVENTS\]](#)

Between the time an element's start tag is parsed and the time either the element's end tag is parsed or the parser detects a well-formedness error, the user agent must act as if the element was in a [stack of open elements](#).

NOTE:

This is used, e.g., by the `<object>` element to avoid instantiating plugins before the `<param>` element children have been parsed.

This specification provides the following additional information that user agents should use when retrieving an external entity: the public identifiers given in the following list all correspond to [the URL given by this link](#). (This URL is a DTD containing the [entity declarations](#) for the names listed in the [§8.5 Named character references](#) section.) [\[XML\]](#)

- `!DOCTYPE HTML 1.0 Transitional//EN`
- `!DOCTYPE HTML 1.1//EN`
- `!DOCTYPE HTML 1.0 Strict//EN`
- `!DOCTYPE HTML 1.0 Frameset//EN`
- `!DOCTYPE HTML Basic 1.0//EN`
- `!DOCTYPE HTML 1.1 plus MathML 2.0//EN`
- `!DOCTYPE HTML 1.1 plus MathML 2.0 plus SVG 1.1//EN`
- `!DOCTYPE MathML 2.0//EN`
- `!DOCTYPE HTML Mobile 1.0//EN`

Furthermore, user agents should attempt to retrieve the above external entity's content when one of the above public identifiers is used, and should not attempt to retrieve any other external entity's content.

NOTE:

This is not strictly a [violation](#) of the XML specification, but it does contradict the spirit of the XML specification's requirements. This is motivated by a desire for user agents to all handle entities in an interoperable fashion without requiring any network access for handling external subsets. [\[XML\]](#)

XML parsers can be invoked with **XML scripting support enabled or disabled**. Except where otherwise specified, XML parsers are invoked with [XML scripting support enabled](#).

When an [XML parser](#) with [XML scripting support enabled](#) creates a `<script>` element, it must be marked as being ["parser-inserted"](#) and its ["non-blocking"](#) flag must be unset. If the parser was originally created for the [XML fragment parsing algorithm](#), then the element must be marked as ["already started"](#) also. When the element's end tag is subsequently parsed, the user agent must [perform a microtask checkpoint](#), and then prepare the `<script>` element. If this causes there to be a [pending parsing-blocking script](#), then the user agent must run the following steps:

1. Block this instance of the [XML parser](#), such that the [event loop](#) will not run [tasks](#) that invoke it.
2. [Spin the event loop](#) until the parser's Document [has no style sheet that is blocking scripts](#) and the [pending parsing-blocking script's "ready to be parser-executed"](#) flag is set.
3. Unblock this instance of the [XML parser](#), such that [tasks](#) that invoke it can again be run.
4. [Execute](#) the [pending parsing-blocking script](#).
5. There is no longer a [pending parsing-blocking script](#).

NOTE:

Since the `document.write()` API is not available for [XML documents](#), much of the complexity in the [HTML parser](#) is not needed in the [XML parser](#).

NOTE:

When the [XML parser](#) has [XML scripting support disabled](#), none of this happens.

When an [XML parser](#) would append a node to a `<template>` element, it must instead append it to the `<template>` element's [template contents](#) (a DocumentFragment node).

NOTE:

This is a [willful violation](#) of the XML specification; unfortunately, XML is not formally extensible in the manner that is needed for template processing. [\[XML\]](#)

When an [XML parser](#) creates a Node object, its [node document](#) must be set to the [node document](#) of the node into which the newly created node is to be inserted.

Certain algorithms in this specification **spoon-feed the parser** characters one string at a time. In such cases, the [XML parser](#) must act as it would have if faced with a single string consisting of the concatenation of all those characters.

When an [XML parser](#) reaches the end of its input, it must [stop parsing](#), following the same rules as the [HTML parser](#). An [XML parser](#) can also be [aborted](#), which must again be done in the same way as for an [HTML parser](#).

For the purposes of conformance checkers, if a resource is determined to be in [the XHTML syntax](#), then it is an [XML document](#).

§ 9.3. Serializing XHTML fragments

The **XML fragment serialization algorithm** for a Document or [Element](#) node either returns a fragment of XML that represents that node or throws an exception.

For Documents, the algorithm must return a string in the form of a [document entity](#), if none of the error cases below apply.

For [Elements](#), the algorithm must return a string in the form of an [internal general parsed entity](#), if none of the error cases below apply.

In both cases, the string returned must be XML namespace-well-formed and must be an isomorphic serialization of all of that node's [relevant child nodes](#), in [tree order](#). User agents may adjust prefixes and namespace declarations in the serialization (and indeed might be forced to do so in some cases to obtain namespace-well-formed XML). User agents may use a combination of regular text and character references to represent Text nodes in the DOM.

A node's **relevant child nodes** are those that apply given the following rules:

For [<template>](#) elements

The [relevant child nodes](#) are the child nodes of the template element's [template contents](#), if any.

For all other nodes

The [relevant child nodes](#) are the child nodes of node itself, if any.

For [Elements](#), if any of the elements in the serialization are in no namespace, the default namespace in scope for those elements must be explicitly declared as the empty string. (This doesn't apply in the Document case.) [\[XML\]](#) [\[XML-NAMES\]](#)

For the purposes of this section, an internal general parsed entity is considered XML namespace-well-formed if a document consisting of an element with no namespace declarations whose contents are the internal general parsed entity would itself be XML namespace-well-formed.

If any of the following error cases are found in the DOM subtree being serialized, then the algorithm must throw an `InvalidStateError` exception instead of returning a string:

- A Document node with no child element nodes.
- A `DocumentType` node that has an external subset public identifier that contains characters that are not matched by the XML PubidChar production. [\[XML\]](#)
- A `DocumentType` node that has an external subset system identifier that contains both a U+0022 QUOTATION MARK (") and a U+0027 APOSTROPHE (') or that contains characters that are not matched by the XML Char production. [\[XML\]](#)
- A node with a local name containing a U+003A COLON (:).
- A node with a local name that does not match the XML [Name](#) production. [\[XML\]](#)
- An `Attr` node with no namespace whose local name is the lowercase string "xmlns". [\[XML-NAMES\]](#)
- An [Element](#) node with two or more attributes with the same local name and namespace.
- An `Attr` node, Text node, Comment node, or `ProcessingInstruction` node whose data contains characters that are not matched by the XML Char production. [\[XML\]](#)
- A Comment node whose data contains two adjacent U+002D HYPHEN-MINUS characters (-) or ends with such a character.

- A `ProcessingInstruction` node whose target name is an [ASCII case-insensitive](#) match for the string `"xml"`.
- A `ProcessingInstruction` node whose target name contains a U+003A COLON (`:`).
- A `ProcessingInstruction` node whose data contains the string `"?>"`.

NOTE:

These are the only ways to make a DOM unserialisable. The DOM enforces all the other XML constraints; for example, trying to append two elements to a `Document` node will throw a `HierarchyRequestError` exception.

§ 9.4. Parsing XHTML fragments

The **XML fragment parsing algorithm** either returns a `Document` or throws a `"SyntaxError" DOMException`. Given a string `input` and a context element `context`, the algorithm is as follows:

1. Create a new [XML parser](#).
2. [Feed the parser](#) just created the string corresponding to the start tag of the `context` element, declaring all the namespace prefixes that are in scope on that element in the DOM, as well as declaring the default namespace (if any) that is in scope on that element in the DOM.

A namespace prefix is in scope if the DOM `lookupNamespaceURI()` method on the element would return a non-null value for that prefix.

The default namespace is the namespace for which the DOM `isDefaultNamespace()` method on the element would return true.

NOTE:

No DOCTYPE is passed to the parser, and therefore no external subset is referenced, and therefore no entities will be recognized.

3. [Feed the parser](#) just created the string `input`.
4. [Feed the parser](#) just created the string corresponding to the end tag of the `context` element.
5. If there is an XML well-formedness or XML namespace well-formedness error, then throw a `"SyntaxError" DOMException` and abort these steps.
6. If the root element of the resulting [Document](#) has any sibling nodes, then throw a `"SyntaxError" DOMException` and abort these steps.
7. Return the child nodes of the root element of the resulting [Document](#), in [tree order](#).

§ 10. Rendering

User agents are not required to present HTML documents in any particular way. However, this section provides a set of suggestions for rendering HTML documents that, if followed, are likely to lead to a user experience that closely resembles the experience intended by the documents' authors. So as to avoid confusion regarding the normativity of this section, RFC2119 terms have not been used. Instead, the term "expected" is used to indicate behavior that will lead to this experience. For the purposes of conformance for user agents designated as [supporting the suggested default rendering](#), the term "expected" in this section has the same conformance implications as the RFC2119-defined term "must".

§ 10.1. Introduction

In general, user agents are expected to support CSS, and many of the suggestions in this section are expressed in CSS terms. User agents that use other presentation mechanisms can derive their expected behavior by translating from the CSS rules given in this section.

In the absence of style-layer rules to the contrary (e.g., author style sheets), user agents are expected to render an element so

that it conveys to the user the meaning that the element [represents](#), as described by this specification.

The suggestions in this section generally assume a visual output medium with a resolution of 96dpi or greater, but HTML is intended to apply to multiple media (it is a *media-independent* language). User agent implementors are encouraged to adapt the suggestions in this section to their target media.



An element is **being rendered** if it has any associated CSS layout boxes, SVG layout boxes, or some equivalent in other styling languages.

NOTE:

Just being off-screen does not mean the element is not [being rendered](#). The presence of the `hidden` attribute normally means the element is not [being rendered](#), though this might be overridden by the style sheets.



User agents that do not honor author-level CSS style sheets are nonetheless expected to act as if they applied the CSS rules given in these sections in a manner consistent with this specification and the relevant CSS and Unicode specifications.

[\[CSS-2015\]](#) [\[UNICODE\]](#) [\[BIDI\]](#)

NOTE:

This is especially important for issues relating to the [‘display’](#), [‘unicode-bidi’](#), and [‘direction’](#) properties.

§ 10.2. The CSS user agent style sheet and presentational hints

The CSS rules given in these subsections are, except where otherwise specified, expected to be used as part of the user-agent level style sheet defaults for all documents that contain [html elements](#).

Some rules are intended for the author-level zero-specificity presentational hints part of the CSS cascade; these are explicitly called out as **presentational hints**.

Some of the rules regarding left and right margins are given here as appropriate for elements whose [‘direction’](#) property is [‘ltr’](#), and are expected to be flipped around on elements whose [‘direction’](#) property is [‘rtl’](#). These are marked "LTR-specific".

NOTE:

These markings only affect the handling of attribute values, not attribute names or element names.



When the text below says that an attribute *attribute* on an element *element* **maps to the pixel length property** (or properties) *properties*, it means that if *element* has an attribute *attribute* set, and parsing that attribute's value using the [rules for parsing non-negative integers](#) doesn't generate an error, then the user agent is expected to use the parsed value as a pixel length for a [presentational hint](#) for *properties*.

When the text below says that an attribute *attribute* on an element *element* **maps to the dimension property** (or properties) *properties*, it means that if *element* has an attribute *attribute* set, and parsing that attribute's value using the [rules for parsing dimension values](#) doesn't generate an error, then the user agent is expected to use the parsed dimension as the value for a [presentational hint](#) for *properties*, with the value given as a pixel length if the dimension was a length, and with the value given as a percentage if the dimension was a percentage.

When the text below says that an attribute *attribute* on an element *element* **maps to the dimension property (ignoring zero)** (or properties) *properties*, it means that if *element* has an attribute *attribute* set, and parsing that attribute's value using the [rules for parsing non-zero dimension values](#) doesn't generate an error, then the user agent is expected to use the parsed dimension as the value for a [presentational hint](#) for *properties*, with the value given as a pixel length if the dimen-

sion was a length, and with the value given as a percentage if the dimension was a percentage.

When a user agent is to **align descendants** of a node, the user agent is expected to align only those descendants that have both their `'margin-left'` and `'margin-right'` properties computing to a value other than `'auto'`, that are over-constrained and that have one of those two margins with a used value forced to a greater value, and that do not themselves have an applicable `align` attribute. When multiple elements are to [align](#) a particular descendant, the most deeply nested such element is expected to override the others. Aligned elements are expected to be aligned by having the used values of their left and right margins be set accordingly.

§ 10.3. Non-replaced elements

§ 10.3.1. Hidden elements

```
@namespace url(http://www.w3.org/1999/xhtml);

[hidden], area, base, basefont, datalist, head, link, menu[type=context i], meta,
noembed, noframes, param, rp, script, source, style, template, track, title {
  display: none;
}

embed[hidden] { display: inline; height: 0; width: 0; }

input[type=hidden i] { display none ! important; }

@media (scripting) {
  noscript { display: none !important; }
}
```

§ 10.3.2. The page

```
@namespace url(http://www.w3.org/1999/xhtml);

html, body { display: block; }
```

For each property in the table below, given a `<body>` element, the first attribute that exists [maps to the pixel length property](#) on the `<body>` element. If none of the attributes for a property are found, or if the value of the attribute that was found cannot be parsed successfully, then, then a default value of 8px is expected to be used for that property instead.

Property	Source
‘margin-top’	<code><body></code> element’s <code>marginheight</code> attribute
	The <code><body></code> element’s container frame element ’s <code>marginheight</code> attribute
	<code><body></code> element’s <code>topmargin</code> attribute
‘margin-right’	<code><body></code> element’s <code>marginwidth</code> attribute
	The <code><body></code> element’s container frame element ’s <code>marginwidth</code> attribute
	<code><body></code> element’s <code>rightmargin</code> attribute
‘margin-bottom’	<code><body></code> element’s <code>marginheight</code> attribute
	The <code><body></code> element’s container frame element ’s <code>marginheight</code> attribute
	<code><body></code> element’s <code>bottommargin</code> attribute
‘margin-left’	<code><body></code> element’s <code>marginwidth</code> attribute
	The <code><body></code> element’s container frame element ’s <code>marginwidth</code> attribute
	<code><body></code> element’s <code>leftmargin</code> attribute

If the `<body>` element’s [node document](#)’s [browsing context](#) is a [nested browsing context](#), and the [browsing context container](#) of that [nested browsing context](#) is a frame or `<iframe>` element, then the **container frame element** of the `<body>` element is

that frame or `<iframe>` element. Otherwise, there is no [container frame element](#).

⚠Warning! The above requirements imply that a page can change the margins of another page (including one from another [origin](#)) using, for example, an `<iframe>`. This is potentially a security risk, as it might in some cases allow an attack to contrive a situation in which a page is rendered not as the author intended, possibly for the purposes of phishing or otherwise misleading the user.



If a [Document](#) is in a [nested browsing context](#), it is expected to be positioned and sized to fit inside the content box of its [browsing context container](#). If a [browsing context](#) is not [being rendered](#), it is expected to have a [viewport](#) with zero width and zero height.

If the [Document](#) is in a [nested browsing context](#), and the [browsing context container](#) of that [nested browsing context](#) is a `<frame>` or `<iframe>` element, and that element has a `scrolling` attribute, and that attribute's value is an [ASCII case-insensitive](#) match for the string "off", "noscroll", or "no", then the user agent is expected to prevent any scroll bars from being shown for the [viewport](#) of the [nested browsing context](#), regardless of the `'overflow'` property that applies to that [viewport](#).



When a `<body>` element has a `background` attribute set to a non-empty value, the new value is expected to be [parsed](#) relative to the element's [node document](#), and if this is successful, the user agent is expected to treat the attribute as a [presentational hint](#) setting the element's `'background-image'` property to the [resulting URL string](#).

When a `<body>` element has a `bgcolor` attribute set, the new value is expected to be parsed using the [rules for parsing a legacy color value](#), and if that does not return an error, the user agent is expected to treat the attribute as a [presentational hint](#) setting the element's `'background-color'` property to the resulting color.

When a `<body>` element has a `text` attribute, its value is expected to be parsed using the [rules for parsing a legacy color value](#), and if that does not return an error, the user agent is expected to treat the attribute as a [presentational hint](#) setting the element's `'color'` property to the resulting color.

When a `<body>` element has a `link` attribute, its value is expected to be parsed using the [rules for parsing a legacy color value](#), and if that does not return an error, the user agent is expected to treat the attribute as a [presentational hint](#) setting the `'color'` property of any element in the Document matching the `'link'` pseudo-class to the resulting color.

When a `<body>` element has a `vlink` attribute, its value is expected to be parsed using the [rules for parsing a legacy color value](#), and if that does not return an error, the user agent is expected to treat the attribute as a [presentational hint](#) setting the `'color'` property of any element in the Document matching the `'visited'` pseudo-class to the resulting color.

When a `<body>` element has an `alink` attribute, its value is expected to be parsed using the [rules for parsing a legacy color value](#), and if that does not return an error, the user agent is expected to treat the attribute as a [presentational hint](#) setting the `'color'` property of any element in the Document matching the `'active'` pseudo-class and either the `'link'` pseudo-class or the `'visited'` pseudo-class to the resulting color.

§ 10.3.3. Flow content


```

@namespace url(http://www.w3.org/1999/xhtml);

address, blockquote, center, div, figure, figcaption, footer, form, header, hr,
legend, listing, main, p, plaintext, pre, summary, xmp {
  display: block;
}

blockquote, figure, listing, p, plaintext, pre, xmp {
  margin-top: 1em; margin-bottom: 1em;
}

blockquote, figure { margin-left: 40px; margin-right: 40px; }

address { font-style: italic; }
listing, plaintext, pre, xmp {
  font-family: monospace; white-space: pre;
}

```

The following rules are also expected to apply, as [presentational hints](#):

```

@namespace url(http://www.w3.org/1999/xhtml);

pre[wrap] { white-space: pre-wrap; }

```

In [quirks mode](#), the following rules are also expected to apply:

```

@namespace url(http://www.w3.org/1999/xhtml);

form { margin-bottom: 1em; }

```



The `<center>` element, and the `<div>` element when it has an [align](#) attribute whose value is an [ASCII case-insensitive](#) match for either the string "center" or the string "middle", are expected to center text within themselves, as if they had their [‘text-align’](#) property set to [‘center’](#) in a [presentational hint](#), and to [align descendants](#) to the center.

The `<div>` element, when it has an [align](#) attribute whose value is an [ASCII case-insensitive](#) match for the string "left", is expected to left-align text within itself, as if it had its [‘text-align’](#) property set to [‘left’](#) in a [presentational hint](#), and to [align descendants](#) to the left.

The `<div>` element, when it has an [align](#) attribute whose value is an [ASCII case-insensitive](#) match for the string "right", is expected to right-align text within itself, as if it had its [‘text-align’](#) property set to [‘right’](#) in a [presentational hint](#), and to [align descendants](#) to the right.

The `<div>` element, when it has an [align](#) attribute whose value is an [ASCII case-insensitive](#) match for the string "justify", is expected to full-justify text within itself, as if it had its [‘text-align’](#) property set to [‘justify’](#) in a [presentational hint](#), and to [align descendants](#) to the left.

§ 10.3.4. Phrasing content

```

@namespace url(http://www.w3.org/1999/xhtml);

cite, dfn, em, i, var { font-style: italic; }
b, strong { font-weight: bolder; }
code, kbd, samp, tt { font-family: monospace; }
big { font-size: larger; }
small { font-size: smaller; }

sub { vertical-align: sub; }
sup { vertical-align: super; }
sub, sup { line-height: normal; font-size: smaller; }

ruby { display: ruby; }
rb { display: ruby-base; white-space: nowrap; }
rt {
    display: ruby-text;
    white-space: nowrap;
    font-size: 50%;
    font-variant-east-asian: ruby;
    text-emphasis: none;
}
rbc { display: ruby-base-container; }
rtc { display: ruby-text-container; }
ruby, rb, rt, rbc, rtc { unicode-bidi: isolate; }

:link { color: #0000EE; }
:visited { color: #551A8B; }
:link:active, :visited:active { color: #FF0000; }
:link, :visited { text-decoration: underline; cursor: pointer; }
a:link[rel~=help], a:visited[rel~=help],
area:link[rel~=help], area:visited[rel~=help] { cursor: help; }

:focus { outline: auto; }

mark { background: yellow; color: black; } /* this color is just a suggestion and can be
changed based on implementation feedback */

abbr[title], acronym[title] { text-decoration: dotted underline; }
ins, u { text-decoration: underline; }
del, s, strike { text-decoration: line-through; }
blink { text-decoration: blink; }

q::before { content: open-quote; }
q::after { content: close-quote; }

br { display-outside: newline; } /* this also has bidi implications */
nobr { white-space: nowrap; }
wbr { display-outside: break-opportunity; } /* this also has bidi implications */
nobr wbr { white-space: normal; }

```

The following rules are also expected to apply, as [presentational hints](#):

```

@namespace url(http://www.w3.org/1999/xhtml);

br[clear=left i] { clear: left; }
br[clear=right i] { clear: right; }
br[clear=all i], br[clear=both i] { clear: both; }

```

User agents that do not support correct ruby rendering are expected to render parentheses around the text of `<rt>` elements in the absence of `<rp>` elements. [\[CSS3-RUBY\]](#)



User agents are expected to support the ‘[clear](#)’ property on inline elements (in order to render [
](#) elements with `clear` attributes) in the manner described in the non-normative note to this effect in CSS2.1.

The initial value for the ‘[color](#)’ property is expected to be black. The initial value for the ‘[background-color](#)’ property is expected to be ‘[transparent](#)’. The canvas’ background is expected to be white.



When a [](#) element has a `color` attribute, its value is expected to be parsed using the [rules for parsing a legacy color value](#), and if that does not return an error, the user agent is expected to treat the attribute as a [presentational hint](#) setting the element’s ‘[color](#)’ property to the resulting color.

The [](#) element is expected to override the color of any text decoration that spans the text of the element to the used value of the element’s ‘[color](#)’ property.

When a [](#) element has a `face` attribute, the user agent is expected to treat the attribute as a [presentational hint](#) setting the element’s ‘[font-family](#)’ property to the attribute’s value.

When a [](#) element has a `size` attribute, the user agent is expected to use the following steps, known as the **rules for parsing a legacy font size**, to treat the attribute as a [presentational hint](#) setting the element’s ‘[font-size](#)’ property:

1. Let *input* be the attribute’s value.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
3. [Skip whitespace](#).
4. If *position* is past the end of *input*, there is no [presentational hint](#). Abort these steps.
5. If the character at *position* is a U+002B PLUS SIGN character (+), then let *mode* be *relative-plus*, and advance *position* to the next character. Otherwise, if the character at *position* is a U+002D HYPHEN-MINUS character (-), then let *mode* be *relative-minus*, and advance *position* to the next character. Otherwise, let *mode* be *absolute*.
6. [Collect a sequence of characters](#) that are [ASCII digits](#), and let the resulting sequence be *digits*.
7. If *digits* is the empty string, there is no [presentational hint](#). Abort these steps.
8. Interpret *digits* as a base-ten integer. Let *value* be the resulting number.
9. If *mode* is *relative-plus*, then increment *value* by 3. If *mode* is *relative-minus*, then let *value* be the result of subtracting *value* from 3.
10. If *value* is greater than 7, let it be 7.
11. If *value* is less than 1, let it be 1.
12. Set ‘[font-size](#)’ to the keyword corresponding to the value of *value* according to the following table:

<i>value</i>	‘ font-size ’ keyword	Notes
1	x-small	
2	small	
3	medium	
4	large	
5	x-large	
6	xx-large	
7	xxx-large	<i>see below</i>

The “xxx-large” value is a non-CSS value used here to indicate a font size 50% larger than “xx-large”.

§ 10.3.5. Bidirectional text

```
@namespace url(http://www.w3.org/1999/xhtml);

[dir]:dir(ltr), bdi:dir(ltr), input[type=tel i]:dir(ltr) { direction: ltr; }
[dir]:dir(rtl), bdi:dir(rtl) { direction: rtl; }

address, blockquote, center, div, figure, figcaption, footer, form, header, hr,
legend, listing, main, p, plaintext, pre, summary, xmp, article, aside, h1, h2,
h3, h4, h5, h6, nav, section, table, caption, colgroup, col, thead,
tbody, tfoot, tr, td, th, dir, dd, dl, dt, menu, ol, ul, li, bdi, output,
[dir=ltr i], [dir=rtl i], [dir=auto i] {
  unicode-bidi: isolate;
}

bdo, bdo[dir] { unicode-bidi: isolate-override; }

input[dir=auto i]:matches([type=search i], [type=tel i], [type=url i],
[type=email i]), textarea[dir=auto i], pre[dir=auto i] {
  unicode-bidi: plaintext;
}
/* see prose for input elements whose type attribute is in the Text state */

/* the rules setting the 'content' property on br and wbr elements also has bidi implications */
```

When an `<input>` element's `dir` attribute is in the [auto](#) state and its `type` attribute is in the [Text](#) state, then the user agent is expected to act as if it had a user-agent-level style sheet rule setting the `'unicode-bidi'` property to `'plaintext'`.

Input fields (i.e., `<textarea>` elements, and `<input>` elements when their `type` attribute is in the [Text](#), [Search](#), [Telephone](#), [URL](#), or [E-mail](#) state) are expected to present an editing user interface with a directionality that matches the element's `'direction'` property.

When the document's character encoding is [ISO-8859-8](#), the following rules are additionally expected to apply, following those above: [\[ENCODING\]](#)

```
@namespace url(http://www.w3.org/1999/xhtml);

address, blockquote, center, div, figure, figcaption, footer, form, header, hr,
legend, listing, main, p, plaintext, pre, summary, xmp, article, aside, h1, h2,
h3, h4, h5, h6, nav, section, table, caption, colgroup, col, thead,
tbody, tfoot, tr, td, th, dir, dd, dl, dt, menu, ol, ul, li, [dir=ltr i],
[dir=rtl i], [dir=auto i], *|* {
  unicode-bidi: bidi-override;
}

input:not([type=submit i]):not([type=reset i]):not([type=button i]),
textarea, keygen {
  unicode-bidi: normal;
}
```

§ 10.3.6. Quotes

This block is automatically generated from the Unicode Common Locale Data Repository. [\[CLDR\]](#)

User agents are expected to use either the block below (which will be regularly updated) or to automatically generate their own copy directly from the source material. The language codes are derived from the CLDR file names. The quotes are derived from the `delimiter` blocks, with fallback handled as specified in the CLDR documentation.

```
@namespace url(http://www.w3.org/1999/xhtml);
```

```

:root { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(af),      :not(:lang(af)) > :lang(af) { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(agq),      :not(:lang(agq)) > :lang(agq) { quotes: '\201e' '\201d' '\201a'
'\2019' } /* „ ” , ’ */
:root:lang(ak),        :not(:lang(ak)) > :lang(ak) { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(am),        :not(:lang(am)) > :lang(am) { quotes: '\00ab' '\00bb' '\2039'
'\203a' } /* « » ‹ › */
:root:lang(ar),        :not(:lang(ar)) > :lang(ar) { quotes: '\201d' '\201c' '\2019'
'\2018' } /* ” “ ” ‘ */
:root:lang(asa),        :not(:lang(asa)) > :lang(asa) { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(az-Cyrl),   :not(:lang(az-Cyrl)) > :lang(az-Cyrl) { quotes: '\00ab' '\00bb' '\2039'
'\203a' } /* « » ‹ › */
:root:lang(bas),        :not(:lang(bas)) > :lang(bas) { quotes: '\00ab' '\00bb' '\201e'
'\201c' } /* « » „ “ */
:root:lang(bem),        :not(:lang(bem)) > :lang(bem) { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(bez),        :not(:lang(bez)) > :lang(bez) { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(bg),        :not(:lang(bg)) > :lang(bg) { quotes: '\201e' '\201c' '\201a'
'\2018' } /* „ ” “ , ‘ */
:root:lang(bm),        :not(:lang(bm)) > :lang(bm) { quotes: '\00ab' '\00bb' '\201c'
'\201d' } /* « » “ ” */
:root:lang(bn),        :not(:lang(bn)) > :lang(bn) { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(br),        :not(:lang(br)) > :lang(br) { quotes: '\00ab' '\00bb' '\2039'
'\203a' } /* « » ‹ › */
:root:lang(brx),        :not(:lang(brx)) > :lang(brx) { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(bs-Cyrl),   :not(:lang(bs-Cyrl)) > :lang(bs-Cyrl) { quotes: '\201e' '\201c' '\201a'
'\2018' } /* „ ” “ , ‘ */
:root:lang(ca),        :not(:lang(ca)) > :lang(ca) { quotes: '\201c' '\201d' '\00ab'
'\00bb' } /* “ ” « » */
:root:lang(cgg),        :not(:lang(cgg)) > :lang(cgg) { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(chr),        :not(:lang(chr)) > :lang(chr) { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(cs),        :not(:lang(cs)) > :lang(cs) { quotes: '\201e' '\201c' '\201a'
'\2018' } /* „ ” “ , ‘ */
:root:lang(da),        :not(:lang(da)) > :lang(da) { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(dav),        :not(:lang(dav)) > :lang(dav) { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(de),        :not(:lang(de)) > :lang(de) { quotes: '\201e' '\201c' '\201a'
'\2018' } /* „ ” “ , ‘ */
:root:lang(de-CH),     :not(:lang(de-CH)) > :lang(de-CH) { quotes: '\00ab' '\00bb' '\2039'
'\203a' } /* « » ‹ › */
:root:lang(dje),        :not(:lang(dje)) > :lang(dje) { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(dua),        :not(:lang(dua)) > :lang(dua) { quotes: '\00ab' '\00bb' '\2018'
'\2019' } /* « » ‘ ’ */
:root:lang(dyو),        :not(:lang(dyو)) > :lang(dyو) { quotes: '\00ab' '\00bb' '\201c'
'\201d' } /* « » “ ” */
:root:lang(dz),        :not(:lang(dz)) > :lang(dz) { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(ebu),        :not(:lang(ebu)) > :lang(ebu) { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(ee),        :not(:lang(ee)) > :lang(ee) { quotes: '\201c' '\201d' '\2018'

```

```
'\2019' } /* " " ' ' */
:root:lang(el),      :not(:lang(el)) > :lang(el)      { quotes: '\00ab' '\00bb' '\201c'
'\201d' } /* « » " " */
:root:lang(en),      :not(:lang(en)) > :lang(en)      { quotes: '\201c' '\201d' '\2018'
'\2019' } /* " " ' ' */
:root:lang(es),      :not(:lang(es)) > :lang(es)      { quotes: '\201c' '\201d' '\00ab'
'\00bb' } /* " " « » */
:root:lang(et),      :not(:lang(et)) > :lang(et)      { quotes: '\201e' '\201c' '\201a'
'\2018' } /* „ „ ‚ ‘ */
:root:lang(eu),      :not(:lang(eu)) > :lang(eu)      { quotes: '\201c' '\201d' '\00ab'
'\00bb' } /* " " « » */
:root:lang(ewo),      :not(:lang(ewo)) > :lang(ewo)    { quotes: '\00ab' '\00bb' '\201c'
'\201d' } /* « » " " */
:root:lang(fa),      :not(:lang(fa)) > :lang(fa)      { quotes: '\00ab' '\00bb' '\2039'
'\203a' } /* « » < > */
:root:lang(ff),      :not(:lang(ff)) > :lang(ff)      { quotes: '\201e' '\201d' '\201a'
'\2019' } /* „ „ ‚ ’ */
:root:lang(fi),      :not(:lang(fi)) > :lang(fi)      { quotes: '\201d' '\201d' '\2019'
'\2019' } /* ” ” ’ ’ */
:root:lang(fr),      :not(:lang(fr)) > :lang(fr)      { quotes: '\00ab' '\00bb' '\00ab'
'\00bb' } /* « » « » */
:root:lang(fr-CA),    :not(:lang(fr-CA)) > :lang(fr-CA) { quotes: '\00ab' '\00bb' '\2039'
'\203a' } /* « » < > */
:root:lang(fr-CH),    :not(:lang(fr-CH)) > :lang(fr-CH) { quotes: '\00ab' '\00bb' '\2039'
'\203a' } /* « » < > */
:root:lang(gsw),      :not(:lang(gsw)) > :lang(gsw)    { quotes: '\00ab' '\00bb' '\2039'
'\203a' } /* « » < > */
:root:lang(gu),      :not(:lang(gu)) > :lang(gu)      { quotes: '\201c' '\201d' '\2018'
'\2019' } /* " " ' ' */
:root:lang(guz),      :not(:lang(guz)) > :lang(guz)    { quotes: '\201c' '\201d' '\2018'
'\2019' } /* " " ' ' */
:root:lang(ha),      :not(:lang(ha)) > :lang(ha)      { quotes: '\201c' '\201d' '\2018'
'\2019' } /* " " ' ' */
:root:lang(he),      :not(:lang(he)) > :lang(he)      { quotes: '\0022' '\0022' '\0027'
'\0027' } /* " " ' ' */
:root:lang(hi),      :not(:lang(hi)) > :lang(hi)      { quotes: '\201c' '\201d' '\2018'
'\2019' } /* " " ' ' */
:root:lang(hr),      :not(:lang(hr)) > :lang(hr)      { quotes: '\201e' '\201c' '\201a'
'\2018' } /* „ „ ‚ ‘ */
:root:lang(hu),      :not(:lang(hu)) > :lang(hu)      { quotes: '\201e' '\201d' '\00bb'
'\00ab' } /* „ „ ” » */
:root:lang(id),      :not(:lang(id)) > :lang(id)      { quotes: '\201c' '\201d' '\2018'
'\2019' } /* " " ' ' */
:root:lang(ig),      :not(:lang(ig)) > :lang(ig)      { quotes: '\201c' '\201d' '\2018'
'\2019' } /* " " ' ' */
:root:lang(it),      :not(:lang(it)) > :lang(it)      { quotes: '\00ab' '\00bb' '\201c'
'\201d' } /* « » " " */
:root:lang(ja),      :not(:lang(ja)) > :lang(ja)      { quotes: '\300c' '\300d' '\300e'
'\300f' } /* 「 」 『 』 */
:root:lang(jgo),      :not(:lang(jgo)) > :lang(jgo)    { quotes: '\00ab' '\00bb' '\2039'
'\203a' } /* « » < > */
:root:lang(jmc),      :not(:lang(jmc)) > :lang(jmc)    { quotes: '\201c' '\201d' '\2018'
'\2019' } /* " " ' ' */
:root:lang(kab),      :not(:lang(kab)) > :lang(kab)    { quotes: '\00ab' '\00bb' '\201c'
'\201d' } /* « » " " */
:root:lang(kam),      :not(:lang(kam)) > :lang(kam)    { quotes: '\201c' '\201d' '\2018'
'\2019' } /* " " ' ' */
:root:lang(kde),      :not(:lang(kde)) > :lang(kde)    { quotes: '\201c' '\201d' '\2018'
'\2019' } /* " " ' ' */
:root:lang(kea),      :not(:lang(kea)) > :lang(kea)    { quotes: '\201c' '\201d' '\2018'
'\2019' } /* " " ' ' */
:root:lang(khq),      :not(:lang(khq)) > :lang(khq)    { quotes: '\201c' '\201d' '\2018'
'\2019' } /* " " ' ' */
:root:lang(ki),      :not(:lang(ki)) > :lang(ki)      { quotes: '\201c' '\201d' '\2018'
'\2019' } /* " " ' ' */
```

```
'\2019' } /* “ ” ‘ ’ */
:root:lang(kkj),      :not(:lang(kkj)) > :lang(kkj)      { quotes: '\00ab' '\00bb' '\2039'
'\203a' } /* « » ‹ › */
:root:lang(kln),      :not(:lang(kln)) > :lang(kln)      { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(km),       :not(:lang(km)) > :lang(km)        { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(kn),       :not(:lang(kn)) > :lang(kn)        { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(ko),       :not(:lang(ko)) > :lang(ko)        { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(ksb),      :not(:lang(ksb)) > :lang(ksb)      { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(ksf),      :not(:lang(ksf)) > :lang(ksf)      { quotes: '\00ab' '\00bb' '\2018'
'\2019' } /* « » ‹ › */
:root:lang(lag),      :not(:lang(lag)) > :lang(lag)      { quotes: '\201d' '\201d' '\2019'
'\2019' } /* ” ” ” ’ */
:root:lang(lg),       :not(:lang(lg)) > :lang(lg)        { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(ln),       :not(:lang(ln)) > :lang(ln)        { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(lo),       :not(:lang(lo)) > :lang(lo)        { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(lt),       :not(:lang(lt)) > :lang(lt)        { quotes: '\201e' '\201c' '\201e'
'\201c' } /* „ ” „ ” */
:root:lang(lu),       :not(:lang(lu)) > :lang(lu)        { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(luo),      :not(:lang(luo)) > :lang(luo)      { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(luy),      :not(:lang(luy)) > :lang(luy)      { quotes: '\201e' '\201c' '\201a'
'\2018' } /* „ ” , ‘ */
:root:lang(lv),       :not(:lang(lv)) > :lang(lv)        { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(mas),      :not(:lang(mas)) > :lang(mas)      { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(mer),      :not(:lang(mer)) > :lang(mer)      { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(mfe),      :not(:lang(mfe)) > :lang(mfe)      { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(mg),       :not(:lang(mg)) > :lang(mg)        { quotes: '\00ab' '\00bb' '\201c'
'\201d' } /* « » “ ” */
:root:lang(mgo),      :not(:lang(mgo)) > :lang(mgo)      { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(mk),       :not(:lang(mk)) > :lang(mk)        { quotes: '\201e' '\201c' '\201a'
'\2018' } /* „ ” , ‘ */
:root:lang(ml),       :not(:lang(ml)) > :lang(ml)        { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(mr),       :not(:lang(mr)) > :lang(mr)        { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(ms),       :not(:lang(ms)) > :lang(ms)        { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(mua),      :not(:lang(mua)) > :lang(mua)      { quotes: '\00ab' '\00bb' '\201c'
'\201d' } /* « » “ ” */
:root:lang(my),       :not(:lang(my)) > :lang(my)        { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(naq),      :not(:lang(naq)) > :lang(naq)      { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(nb),       :not(:lang(nb)) > :lang(nb)        { quotes: '\00ab' '\00bb' '\2018'
'\2019' } /* « » ‹ › */
:root:lang(nd),       :not(:lang(nd)) > :lang(nd)        { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(nl),       :not(:lang(nl)) > :lang(nl)        { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(nmg),      :not(:lang(nmg)) > :lang(nmg)      { quotes: '\201e' '\201d' '\00ab'
```

```
'\00bb' } /* „ ” « » */
:root:lang(nn),      :not(:lang(nn)) > :lang(nn)      { quotes: '\00ab' '\00bb' '\2018'
'\2019' } /* « » ‘ ’ */
:root:lang(nnh),      :not(:lang(nnh)) > :lang(nnh)    { quotes: '\00ab' '\00bb' '\201c'
'\201d' } /* « » “ ” */
:root:lang(nus),      :not(:lang(nus)) > :lang(nus)    { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(nyn),      :not(:lang(nyn)) > :lang(nyn)    { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(pl),       :not(:lang(pl)) > :lang(pl)      { quotes: '\201e' '\201d' '\00ab'
'\00bb' } /* „ ” « » */
:root:lang(pt),       :not(:lang(pt)) > :lang(pt)      { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(pt-PT),    :not(:lang(pt-PT)) > :lang(pt-PT) { quotes: '\00ab' '\00bb' '\201c'
'\201d' } /* « » “ ” */
:root:lang(rn),       :not(:lang(rn)) > :lang(rn)      { quotes: '\201d' '\201d' '\2019'
'\2019' } /* „ ” ’ ’ */
:root:lang(ro),       :not(:lang(ro)) > :lang(ro)      { quotes: '\201e' '\201d' '\00ab'
'\00bb' } /* „ ” « » */
:root:lang(rof),      :not(:lang(rof)) > :lang(rof)    { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(ru),       :not(:lang(ru)) > :lang(ru)      { quotes: '\00ab' '\00bb' '\201e'
'\201c' } /* « » „ ” */
:root:lang(rw),       :not(:lang(rw)) > :lang(rw)      { quotes: '\00ab' '\00bb' '\2018'
'\2019' } /* « » ‘ ’ */
:root:lang(rwk),      :not(:lang(rwk)) > :lang(rwk)    { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(saq),      :not(:lang(saq)) > :lang(saq)    { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(sbp),      :not(:lang(sbp)) > :lang(sbp)    { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(seh),      :not(:lang(seh)) > :lang(seh)    { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(ses),      :not(:lang(ses)) > :lang(ses)    { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(sg),       :not(:lang(sg)) > :lang(sg)      { quotes: '\00ab' '\00bb' '\201c'
'\201d' } /* « » “ ” */
:root:lang(shi),      :not(:lang(shi)) > :lang(shi)    { quotes: '\00ab' '\00bb' '\201e'
'\201d' } /* « » „ ” */
:root:lang(shi-Latn), :not(:lang(shi-Latn)) > :lang(shi-Latn) { quotes: '\00ab' '\00bb' '\201e'
'\201d' } /* « » „ ” */
:root:lang(si),       :not(:lang(si)) > :lang(si)      { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(sk),       :not(:lang(sk)) > :lang(sk)      { quotes: '\201e' '\201c' '\201a'
'\2018' } /* „ ” , ‘ */
:root:lang(sl),       :not(:lang(sl)) > :lang(sl)      { quotes: '\201e' '\201c' '\201a'
'\2018' } /* „ ” , ‘ */
:root:lang(sn),       :not(:lang(sn)) > :lang(sn)      { quotes: '\201d' '\201d' '\2019'
'\2019' } /* „ ” ’ ’ */
:root:lang(so),       :not(:lang(so)) > :lang(so)      { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(sq),       :not(:lang(sq)) > :lang(sq)      { quotes: '\201e' '\201c' '\201a'
'\2018' } /* „ ” , ‘ */
:root:lang(sr),       :not(:lang(sr)) > :lang(sr)      { quotes: '\201e' '\201c' '\201a'
'\2018' } /* „ ” , ‘ */
:root:lang(sr-Latn), :not(:lang(sr-Latn)) > :lang(sr-Latn) { quotes: '\201e' '\201c' '\201a'
'\2018' } /* „ ” , ‘ */
:root:lang(sv),       :not(:lang(sv)) > :lang(sv)      { quotes: '\201d' '\201d' '\2019'
'\2019' } /* „ ” ’ ’ */
:root:lang(sw),       :not(:lang(sw)) > :lang(sw)      { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(swc),      :not(:lang(swc)) > :lang(swc)    { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(ta),       :not(:lang(ta)) > :lang(ta)      { quotes: '\201c' '\201d' '\2018'
```



```

'\2019' } /* “ ” ‘ ’ */
:root:lang(te),      :not(:lang(te)) > :lang(te)      { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(teo),     :not(:lang(teo)) > :lang(teo)   { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(th),      :not(:lang(th)) > :lang(th)     { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(ti-ER),   :not(:lang(ti-ER)) > :lang(ti-ER) { quotes: '\2018' '\2019' '\201c'
'\201d' } /* ‘ ’ “ ” */
:root:lang(to),      :not(:lang(to)) > :lang(to)     { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(tr),      :not(:lang(tr)) > :lang(tr)     { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(twq),     :not(:lang(twq)) > :lang(twq)   { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(tzm),     :not(:lang(tzm)) > :lang(tzm)   { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(uk),      :not(:lang(uk)) > :lang(uk)     { quotes: '\00ab' '\00bb' '\201e'
'\201c' } /* « » „ “ */
:root:lang(ur),      :not(:lang(ur)) > :lang(ur)     { quotes: '\201d' '\201c' '\2019'
'\2018' } /* ” ” ‘ ’ */
:root:lang(vai),     :not(:lang(vai)) > :lang(vai)   { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(vai-Latn), :not(:lang(vai-Latn)) > :lang(vai-Latn) { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(vi),      :not(:lang(vi)) > :lang(vi)     { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(vun),     :not(:lang(vun)) > :lang(vun)   { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(xh),      :not(:lang(xh)) > :lang(xh)     { quotes: '\2018' '\2019' '\201c'
'\201d' } /* ‘ ’ “ ” */
:root:lang(xog),     :not(:lang(xog)) > :lang(xog)   { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(yav),     :not(:lang(yav)) > :lang(yav)   { quotes: '\00ab' '\00bb' '\00ab'
'\00bb' } /* « » « » */
:root:lang(yo),      :not(:lang(yo)) > :lang(yo)     { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(zh),      :not(:lang(zh)) > :lang(zh)     { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */
:root:lang(zh-Hant), :not(:lang(zh-Hant)) > :lang(zh-Hant) { quotes: '\300c' '\300d' '\300e'
'\300f' } /* 「 」 『 』 */
:root:lang(zu),      :not(:lang(zu)) > :lang(zu)     { quotes: '\201c' '\201d' '\2018'
'\2019' } /* “ ” ‘ ’ */

```

§ 10.3.7. Sections and headings

```
@namespace url(http://www.w3.org/1999/xhtml);
```

```

article, aside, h1, h2, h3, h4, h5, h6, nav, section {
  display: block;
}

```

```

h1 { margin-top: 0.67em; margin-bottom: 0.67em; font-size: 2.00em; font-weight: bold; }
h2 { margin-top: 0.83em; margin-bottom: 0.83em; font-size: 1.50em; font-weight: bold; }
h3 { margin-top: 1.00em; margin-bottom: 1.00em; font-size: 1.17em; font-weight: bold; }
h4 { margin-top: 1.33em; margin-bottom: 1.33em; font-size: 1.00em; font-weight: bold; }
h5 { margin-top: 1.67em; margin-bottom: 1.67em; font-size: 0.83em; font-weight: bold; }
h6 { margin-top: 2.33em; margin-bottom: 2.33em; font-size: 0.67em; font-weight: bold; }

```

In the following CSS block, *x* is shorthand for the following selector: `:matches(article, aside, nav, section)`

```
@namespace url(http://www.w3.org/1999/xhtml);

x h1 { margin-top: 0.83em; margin-bottom: 0.83em; font-size: 1.50em; }
x x h1 { margin-top: 1.00em; margin-bottom: 1.00em; font-size: 1.17em; }
x x x h1 { margin-top: 1.33em; margin-bottom: 1.33em; font-size: 1.00em; }
x x x x h1 { margin-top: 1.67em; margin-bottom: 1.67em; font-size: 0.83em; }
x x x x x h1 { margin-top: 2.33em; margin-bottom: 2.33em; font-size: 0.67em; }
```

NOTE:

The shorthand is used to keep this block at least mildly readable.

§ 10.3.8. Lists

```
@namespace url(http://www.w3.org/1999/xhtml);

dir, dd, dl, dt, menu, ol, ul { display: block; }
li { display: list-item; }

dir, dl, menu, ol, ul { margin-top: 1em; margin-bottom: 1em; }

:matches(dir, dl, menu, ol, ul) :matches(dir, dl, menu, ol, ul) {
  margin-top: 0; margin-bottom: 0;
}

dd { margin-left: 40px; } /* LTR-specific: use 'margin-right' for rtl elements */
dir, menu, ol, ul { padding-left: 40px; } /* LTR-specific: use 'padding-right' for rtl elements
*/

ol { list-style-type: decimal; }

dir, menu, ul {
  list-style-type: disc;
}
:matches(dir, menu, ol, ul) :matches(dir, menu, ul) {
  list-style-type: circle;
}
:matches(dir, menu, ol, ul) :matches(dir, menu, ol, ul) :matches(dir, menu, ul) {
  list-style-type: square;
}
```

The following rules are also expected to apply, as [presentational hints](#):

```
@namespace url(http://www.w3.org/1999/xhtml);

ol[type=1], li[type=1] { list-style-type: decimal; }
ol[type=a], li[type=a] { list-style-type: lower-alpha; }
ol[type=A], li[type=A] { list-style-type: upper-alpha; }
ol[type=i], li[type=i] { list-style-type: lower-roman; }
ol[type=I], li[type=I] { list-style-type: upper-roman; }
ul[type=none i], li[type=none i] { list-style-type: none; }
ul[type=disc i], li[type=disc i] { list-style-type: disc; }
ul[type=circle i], li[type=circle i] { list-style-type: circle; }
ul[type=square i], li[type=square i] { list-style-type: square; }
```

In the above stylesheet, the attribute selectors for the `ol` and `<li>` elements are expected to be treated as [case-sensitive](#).

When rendering `<li>` elements, non-CSS user agents are expected to use the [ordinal value](#) of the `<li>` element to render the counter in the list item marker.

This specification does not yet define the CSS-specific rules for rendering [](#) elements, because CSS doesn't yet provide sufficient hooks for this purpose.

§ 10.3.9. Tables

```

@namespace url(http://www.w3.org/1999/xhtml);

table { display: table; }
caption { display: table-caption; }
colgroup, colgroup[hidden] { display: table-column-group; }
col, col[hidden] { display: table-column; }
thead, thead[hidden] { display: table-header-group; }
tbody, tbody[hidden] { display: table-row-group; }
tfoot, tfoot[hidden] { display: table-footer-group; }
tr, tr[hidden] { display: table-row; }
td, th, td[hidden], th[hidden] { display: table-cell; }

colgroup[hidden], col[hidden], thead[hidden], tbody[hidden],
tfoot[hidden], tr[hidden], td[hidden], th[hidden] {
  visibility: collapse;
}

table {
  box-sizing: border-box;
  border-spacing: 2px;
  border-collapse: separate;
  text-indent: initial;
}
td, th { padding: 1px; }
th { font-weight: bold; }

thead, tbody, tfoot, table > tr { vertical-align: middle; }
tr, td, th { vertical-align: inherit; }

table, td, th { border-color: gray; }
thead, tbody, tfoot, tr { border-color: inherit; }
table[rules=none i], table[rules=groups i], table[rules=rows i],
table[rules=cols i], table[rules=all i], table[frame=void i],
table[frame=above i], table[frame=below i], table[frame=hsides i],
table[frame=lhs i], table[frame=rhs i], table[frame=vsides i],
table[frame=box i], table[frame=border i],
table[rules=none i] > tr > td, table[rules=none i] > tr > th,
table[rules=groups i] > tr > td, table[rules=groups i] > tr > th,
table[rules=rows i] > tr > td, table[rules=rows i] > tr > th,
table[rules=cols i] > tr > td, table[rules=cols i] > tr > th,
table[rules=all i] > tr > td, table[rules=all i] > tr > th,
table[rules=none i] > thead > tr > td, table[rules=none i] > thead > tr > th,
table[rules=groups i] > thead > tr > td, table[rules=groups i] > thead > tr > th,
table[rules=rows i] > thead > tr > td, table[rules=rows i] > thead > tr > th,
table[rules=cols i] > thead > tr > td, table[rules=cols i] > thead > tr > th,
table[rules=all i] > thead > tr > td, table[rules=all i] > thead > tr > th,
table[rules=none i] > tbody > tr > td, table[rules=none i] > tbody > tr > th,
table[rules=groups i] > tbody > tr > td, table[rules=groups i] > tbody > tr > th,
table[rules=rows i] > tbody > tr > td, table[rules=rows i] > tbody > tr > th,
table[rules=cols i] > tbody > tr > td, table[rules=cols i] > tbody > tr > th,
table[rules=all i] > tbody > tr > td, table[rules=all i] > tbody > tr > th,
table[rules=none i] > tfoot > tr > td, table[rules=none i] > tfoot > tr > th,
table[rules=groups i] > tfoot > tr > td, table[rules=groups i] > tfoot > tr > th,
table[rules=rows i] > tfoot > tr > td, table[rules=rows i] > tfoot > tr > th,
table[rules=cols i] > tfoot > tr > td, table[rules=cols i] > tfoot > tr > th,
table[rules=all i] > tfoot > tr > td, table[rules=all i] > tfoot > tr > th {
  border-color: black;
}

```

The following rules are also expected to apply, as [presentational hints](#):

```

@namespace url(http://www.w3.org/1999/xhtml);

table[align=left i] { float: left; }
table[align=right i] { float: right; }
table[align=center i] { margin-left: auto; margin-right: auto; }
thead[align=absmiddle i], tbody[align=absmiddle i], tfoot[align=absmiddle i],
tr[align=absmiddle i], td[align=absmiddle i], th[align=absmiddle i] {
    text-align: center;
}

caption[align=bottom i] { caption-side: bottom; }
p[align=left i], h1[align=left i], h2[align=left i], h3[align=left i],
h4[align=left i], h5[align=left i], h6[align=left i] {
    text-align: left;
}
p[align=right i], h1[align=right i], h2[align=right i], h3[align=right i],
h4[align=right i], h5[align=right i], h6[align=right i] {
    text-align: right;
}
p[align=center i], h1[align=center i], h2[align=center i], h3[align=center i],
h4[align=center i], h5[align=center i], h6[align=center i] {
    text-align: center;
}
p[align=justify i], h1[align=justify i], h2[align=justify i], h3[align=justify i],
h4[align=justify i], h5[align=justify i], h6[align=justify i] {
    text-align: justify;
}
thead[valign=top i], tbody[valign=top i], tfoot[valign=top i],
tr[valign=top i], td[valign=top i], th[valign=top i] {
    vertical-align: top;
}
thead[valign=middle i], tbody[valign=middle i], tfoot[valign=middle i],
tr[valign=middle i], td[valign=middle i], th[valign=middle i] {
    vertical-align: middle;
}
thead[valign=bottom i], tbody[valign=bottom i], tfoot[valign=bottom i],
tr[valign=bottom i], td[valign=bottom i], th[valign=bottom i] {
    vertical-align: bottom;
}
thead[valign=baseline i], tbody[valign=baseline i], tfoot[valign=baseline i],
tr[valign=baseline i], td[valign=baseline i], th[valign=baseline i] {
    vertical-align: baseline;
}

td[nowrap], th[nowrap] { white-space: nowrap; }

table[rules=none i], table[rules=groups i], table[rules=rows i],
table[rules=cols i], table[rules=all i] {
    border-style: hidden;
    border-collapse: collapse;
}
table[border] { border-style: outset; } /* only if border is not equivalent to zero */
table[frame=void i] { border-style: hidden; }
table[frame=above i] { border-style: outset hidden hidden hidden; }
table[frame=below i] { border-style: hidden hidden outset hidden; }
table[frame=hsides i] { border-style: outset hidden outset hidden; }
table[frame=lhs i] { border-style: hidden hidden hidden outset; }
table[frame=rhs i] { border-style: hidden outset hidden hidden; }
table[frame=vsides i] { border-style: hidden outset; }
table[frame=box i], table[frame=border i] { border-style: outset; }

table[border] > tr > td, table[border] > tr > th,
table[border] > tr > td, table[border] > tr > th,

```

```

table[border] > tbody > tr > td, table[border] > tbody > tr > th,
table[border] > tfoot > tr > td, table[border] > tfoot > tr > th {
/* only if border is not equivalent to zero */
  border-width: 1px;
  border-style: inset;
}
table[rules=none i] > tr > td, table[rules=none i] > tr > th,
table[rules=none i] > thead > tr > td, table[rules=none i] > thead > tr > th,
table[rules=none i] > tbody > tr > td, table[rules=none i] > tbody > tr > th,
table[rules=none i] > tfoot > tr > td, table[rules=none i] > tfoot > tr > th,
table[rules=groups i] > tr > td, table[rules=groups i] > tr > th,
table[rules=groups i] > thead > tr > td, table[rules=groups i] > thead > tr > th,
table[rules=groups i] > tbody > tr > td, table[rules=groups i] > tbody > tr > th,
table[rules=groups i] > tfoot > tr > td, table[rules=groups i] > tfoot > tr > th,
table[rules=rows i] > tr > td, table[rules=rows i] > tr > th,
table[rules=rows i] > thead > tr > td, table[rules=rows i] > thead > tr > th,
table[rules=rows i] > tbody > tr > td, table[rules=rows i] > tbody > tr > th,
table[rules=rows i] > tfoot > tr > td, table[rules=rows i] > tfoot > tr > th {
  border-width: 1px;
  border-style: none;
}
table[rules=cols i] > tr > td, table[rules=cols i] > tr > th,
table[rules=cols i] > thead > tr > td, table[rules=cols i] > thead > tr > th,
table[rules=cols i] > tbody > tr > td, table[rules=cols i] > tbody > tr > th,
table[rules=cols i] > tfoot > tr > td, table[rules=cols i] > tfoot > tr > th {
  border-width: 1px;
  border-style: none solid;
}
table[rules=all i] > tr > td, table[rules=all i] > tr > th,
table[rules=all i] > thead > tr > td, table[rules=all i] > thead > tr > th,
table[rules=all i] > tbody > tr > td, table[rules=all i] > tbody > tr > th,
table[rules=all i] > tfoot > tr > td, table[rules=all i] > tfoot > tr > th {
  border-width: 1px;
  border-style: solid;
}

table[rules=groups i] > colgroup {
  border-left-width: 1px;
  border-left-style: solid;
  border-right-width: 1px;
  border-right-style: solid;
}
table[rules=groups i] > thead,
table[rules=groups i] > tbody,
table[rules=groups i] > tfoot {
  border-top-width: 1px;
  border-top-style: solid;
  border-bottom-width: 1px;
  border-bottom-style: solid;
}

table[rules=rows i] > tr, table[rules=rows i] > thead > tr,
table[rules=rows i] > tbody > tr, table[rules=rows i] > tfoot > tr {
  border-top-width: 1px;
  border-top-style: solid;
  border-bottom-width: 1px;
  border-bottom-style: solid;
}

```

In [quirks mode](#), the following rules are also expected to apply:

```
@namespace url(http://www.w3.org/1999/xhtml);

table {
  font-weight: initial;
  font-style: initial;
  font-variant: initial;
  font-size: initial;
  line-height: initial;
  white-space: initial;
  text-align: initial;
}
```



For the purposes of the CSS table model, the `<col>` element is expected to be treated as if it was present as many times as its span attribute [specifies](#).

For the purposes of the CSS table model, the `<colgroup>` element, if it contains no `<col>` element, is expected to be treated as if it had as many such children as its span attribute [specifies](#).

For the purposes of the CSS table model, the `colspan` and `rowspan` attributes on `<td>` and `<th>` elements are expected to [provide](#) the *special knowledge* regarding cells spanning rows and columns.

In [HTML documents](#), the following rules are also expected to apply:

```
@namespace url(http://www.w3.org/1999/xhtml);

:matches(table, thead, tbody, tfoot, tr) > form { display: none !important; }
```



The `<table>` element's `cellspacing` attribute [maps to the pixel length property](#) `'border-spacing'` on the element.

The `<table>` element's `cellpadding` attribute [maps to the pixel length properties](#) `'padding-top'`, `'padding-right'`, `'padding-bottom'`, and `'padding-left'` of any `td` and `<th>` elements that have corresponding [cells](#) in the [table](#) corresponding to the `<table>` element.

The `<table>` element's `hspace` attribute [maps to the dimension properties](#) `'margin-left'` and `'margin-right'` on the `<table>` element.

The `<table>` element's `vspace` attribute [maps to the dimension properties](#) `'margin-top'` and `'margin-bottom'` on the `<table>` element.

The `<table>` element's `height` attribute [maps to the dimension property \(ignoring zero\)](#) `'height'` on the `table` element.

The `<table>` element's `width` attribute [maps to the dimension property \(ignoring zero\)](#) `'width'` on the `table` element.

The `<col>` element's `width` attribute [maps to the dimension property \(ignoring zero\)](#) `'width'` on the `<col>` element.

The `<tr>` element's `height` attribute [maps to the dimension property \(ignoring zero\)](#) `'height'` on the `<tr>` element.

The `td` and `<th>` elements' `height` attributes [map to the dimension property \(ignoring zero\)](#) `'height'` on the element.

The `td` and `<th>` elements' `width` attributes [map to the dimension property \(ignoring zero\)](#) `'width'` on the element.



The `<caption>` element unless specified otherwise below, and the `<thead>`, `<tbody>`, `<tfoot>`, `<tr>`, `<td>`, and `th` elements when they have an `align` attribute whose value is an [ASCII case-insensitive](#) match for either the string "center" or the string "middle", are expected to center text within themselves, as if they had their `'text-align'` property set to `'center'` in a

[presentational hint](#), and to [align descendants](#) to the center.

The `<caption>`, `<thead>`, `<tbody>`, `<tfoot>`, `<tr>`, `<td>`, and `<th>` elements, when they have an `align` attribute whose value is an [ASCII case-insensitive](#) match for the string "left", are expected to left-align text within themselves, as if they had their `'text-align'` property set to 'left' in a [presentational hint](#), and to [align descendants](#) to the left.

The `<caption>`, `<thead>`, `<tbody>`, `<tfoot>`, `<tr>`, `<td>`, and `<th>` elements, when they have an `align` attribute whose value is an [ASCII case-insensitive](#) match for the string "right", are expected to right-align text within themselves, as if they had their `'text-align'` property set to 'right' in a [presentational hint](#), and to [align descendants](#) to the right.

The `<caption>`, `<thead>`, `<tbody>`, `<tfoot>`, `<tr>`, `<td>`, and `<th>` elements, when they have an `align` attribute whose value is an [ASCII case-insensitive](#) match for the string "justify", are expected to full-justify text within themselves, as if they had their `'text-align'` property set to 'justify' in a [presentational hint](#), and to [align descendants](#) to the left.

User agents are expected to have a rule in their user agent stylesheet that matches `<th>` elements that have a parent node whose computed value for the `'text-align'` property is its initial value, whose declaration block consists of just a single declaration that sets the `'text-align'` property to the value 'center'.



When a `<table>`, `<thead>`, `<tbody>`, `<tfoot>`, `<tr>`, `<td>`, or `<th>` element has a `background` attribute set to a non-empty value, the new value is expected to be [parsed](#) relative to the element's [node document](#), and if this is successful, the user agent is expected to treat the attribute as a [presentational hint](#) setting the element's `'background-image'` property to the [resulting URL string](#).

When a `<table>`, `<thead>`, `<tbody>`, `<tfoot>`, `<tr>`, `<td>`, or `<th>` element has a `bgcolor` attribute set, the new value is expected to be parsed using the [rules for parsing a legacy color value](#), and if that does not return an error, the user agent is expected to treat the attribute as a [presentational hint](#) setting the element's `'background-color'` property to the resulting color.

When a `<table>` element has a `bordercolor` attribute, its value is expected to be parsed using the [rules for parsing a legacy color value](#), and if that does not return an error, the user agent is expected to treat the attribute as a [presentational hint](#) setting the element's `'border-top-color'`, `'border-right-color'`, `'border-bottom-color'`, and `'border-left-color'` properties to the resulting color.



The `<table>` element's `border` attribute [maps to the pixel length properties](#) `'border-top-width'`, `'border-right-width'`, `'border-bottom-width'`, `'border-left-width'` on the element. If the attribute is present but parsing the attribute's value using the [rules for parsing non-negative integers](#) generates an error, a default value of 1px is expected to be used for that property instead.

Rules marked "only if border is not equivalent to zero" in the CSS block above is expected to only be applied if the `border` attribute mentioned in the selectors for the rule is not only present but, when parsed using the [rules for parsing non-negative integers](#), is also found to have a value other than zero or to generate an error.



In [quirks mode](#), a `<td>` element or a `<th>` element that has a `nowrap` attribute but also has a `width` attribute whose value, when parsed using the [rules for parsing non-zero dimension values](#), is found to be a length (not an error or a number classified as a percentage), is expected to have a [presentational hint](#) setting the element's `'white-space'` property to 'normal', overriding the rule in the CSS block above that sets it to 'nowrap'.

§ 10.3.10. Margin collapsing quirks

A node is **substantial** if it is a text node that is not [inter-element whitespace](#), or if it is an element node.

A node is **blank** if it is an element that contains no [substantial](#) nodes.

The **elements with default margins** are the following elements: `<blockquote>`, `<div>`, `<dl>`, `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`,

[<h6>](#), [<listing>](#), [<menu>](#), [](#), [<p>](#), [<plaintext>](#), [<pre>](#), [](#), xmp

In [quirks mode](#), any [element with default margins](#) that is the child of a [<body>](#), [<td>](#), or [<th>](#) element and has no [substantial](#) previous siblings is expected to have a user-agent level style sheet rule that sets its [‘margin-top’](#) property to zero.

In [quirks mode](#), any [element with default margins](#) that is the child of a [<body>](#), [<td>](#), or [<th>](#) element, has no [substantial](#) previous siblings, and is [blank](#), is expected to have a user-agent level style sheet rule that sets its [‘margin-bottom’](#) property to zero also.

In [quirks mode](#), any [element with default margins](#) that is the child of a [<td>](#) or [<th>](#) element, has no [substantial](#) following siblings, and is [blank](#), is expected to have a user-agent level style sheet rule that sets its [‘margin-top’](#) property to zero.

In [quirks mode](#), any [<p>](#) element that is the child of a [<td>](#) or [<th>](#) element and has no [substantial](#) following siblings, is expected to have a user-agent level style sheet rule that sets its [‘margin-bottom’](#) property to zero.

§ 10.3.11. Form controls

```
@namespace url(http://www.w3.org/1999/xhtml);

input, select, option, optgroup, button, textarea, keygen {
  text-indent: initial;
}

input:matches([type=radio i], [type=checkbox i], [type=reset i], [type=button i],
[type=submit i], [type=search i]), select, button {
  box-sizing: border-box;
}
```

In [quirks mode](#), the following rules are also expected to apply:

```
@namespace url(http://www.w3.org/1999/xhtml);

input:not([type=image i]), textarea { box-sizing: border-box; }
```

Each kind of form control is also given a specific default binding, as described in subsequent sections, which implements the look and feel of the control.

§ 10.3.12. The [<hr>](#) element

```
@namespace url(http://www.w3.org/1999/xhtml);

hr { color: gray; border-style: inset; border-width: 1px; margin: 0.5em auto; }
```

The following rules are also expected to apply, as [presentational hints](#):

```
@namespace url(http://www.w3.org/1999/xhtml);

hr[align=left] { margin-left: 0; margin-right: auto; }
hr[align=right] { margin-left: auto; margin-right: 0; }
hr[align=center] { margin-left: auto; margin-right: auto; }
hr[color], hr[noshade] { border-style: solid; }
```

If an [<hr>](#) element has either a [color](#) attribute or a [noshade](#) attribute, and furthermore also has a [size](#) attribute, and parsing that attribute’s value using the [rules for parsing non-negative integers](#) doesn’t generate an error, then the user agent is expected to use the parsed value divided by two as a pixel length for [presentational hints](#) for the properties [‘border-top-width’](#), [‘border-right-width’](#), [‘border-bottom-width’](#), and [‘border-left-width’](#) on the element.

Otherwise, if an [<hr>](#) element has neither a [color](#) attribute nor a [noshade](#) attribute, but does have a [size](#) attribute, and pars-

ing that attribute's value using the [rules for parsing non-negative integers](#) doesn't generate an error, then: if the parsed value is one, then the user agent is expected to use the attribute as a [presentational hint](#) setting the element's `'border-bottom-width'` to 0; otherwise, if the parsed value is greater than one, then the user agent is expected to use the parsed value minus two as a pixel length for [presentational hints](#) for the `'height'` property on the element.

The width attribute on an `<hr>` element [maps to the dimension property 'width'](#) on the element.

When an `<hr>` element has a `color` attribute, its value is expected to be parsed using the [rules for parsing a legacy color value](#), and if that does not return an error, the user agent is expected to treat the attribute as a [presentational hint](#) setting the element's `'color'` property to the resulting color.

§ 10.3.13. The `<fieldset>` and `<legend>` elements

```
@namespace url(http://www.w3.org/1999/xhtml);
```

```
fieldset {
  display: block;
  margin-left: 2px; margin-right: 2px;
  border: groove 2px ThreeDFace;
  padding: 0.35em 0.625em 0.75em;
  min-width: min-content;
}

legend {
  padding-left: 2px; padding-right: 2px;
}
```

The `<fieldset>` element is expected to establish a new block formatting context.

If the `<fieldset>` element has a child that matches the conditions in the list below, then the first such child is the `<fieldset>` element's **rendered legend**:

- The child is a `<legend>` element.
- The child is not out-of-flow (e.g., not absolutely positioned or floated).
- The child is generating a box (e.g., it is not `'display:none'`).

A `<fieldset>` element's **rendered legend**, if any, is expected to be rendered over the top border edge of the `<fieldset>` element as a `'block'` box (overriding any explicit `'display'` value). In the absence of an explicit width, the box should shrink-wrap. If the `<legend>` element in question has an `align` attribute, and its value is an [ASCII case-insensitive](#) match for one of the strings in the first column of the following table, then the legend is expected to be rendered horizontally aligned over the border edge in the position given in the corresponding cell on the same row in the second column. If the attribute is absent or has a value that doesn't match any of the cases in the table, then the position is expected to be on the right if the `'direction'` property on this element has a computed value of `'rtl'`, and on the left otherwise.

Attribute value	Alignment position
left	On the left
right	On the right
center	In the middle

§ 10.4. Replaced elements

§ 10.4.1. Embedded content

The `<embed>`, `<iframe>`, and `<video>` elements are expected to be treated as [replaced elements](#).

A `<canvas>` element that [represents embedded content](#) is expected to be treated as a [replaced element](#); the contents of such elements are the element's bitmap, if any, or else a transparent black bitmap with the same [intrinsic dimensions](#) as the element. Other `<canvas>` elements are expected to be treated as ordinary elements in the rendering model.

An `<object>` element that [represents](#) an image, plugin, or [nested browsing context](#) is expected to be treated as a [replaced ele-](#)

ment. Other `<object>` elements are expected to be treated as ordinary elements in the rendering model.

An `<applet>` element that represents a plugin is expected to be treated as a [replaced element](#). Other `<applet>` elements are expected to be treated as ordinary elements in the rendering model.

The `<audio>` element, when it is [exposing a user interface](#), is expected to be treated as a [replaced element](#) about one line high, as wide as is necessary to expose the user agent's user interface features. When an `<audio>` element is not [exposing a user interface](#), the user agent is expected to force its `'display'` property to compute to `'none'`, irrespective of CSS rules.

Whether a `<video>` element is [exposing a user interface](#) is not expected to affect the size of the rendering; controls are expected to be overlaid above the page content without causing any layout changes, and are expected to disappear when the user does not need them.

When a `<video>` element represents a poster frame or frame of video, the poster frame or frame of video is expected to be rendered at the largest size that maintains the aspect ratio of that poster frame or frame of video without being taller or wider than the `<video>` element itself, and is expected to be centered in the `<video>` element.

Any subtitles or captions are expected to be overlaid directly on top of their `<video>` element, as defined by the relevant rendering rules; for WebVTT, those are the [rules for updating the display of WebVTT text tracks](#). [WEBVTT]

When the user agent starts [exposing a user interface](#) for a `<video>` element, the user agent should run the [rules for updating the text track rendering](#) of each of the [text tracks](#) in the `<video>` element's [list of text tracks](#) that are [showing](#) and whose [text track kind](#) is one of subtitles or captions (e.g., for [text tracks](#) based on WebVTT, the [rules for updating the display of WebVTT text tracks](#)). [WEBVTT]

NOTE:

Resizing video and `<canvas>` elements does not interrupt video playback or clear the canvas.



The following CSS rules are expected to apply:

```
@namespace url(http://www.w3.org/1999/xhtml);

video { object-fit: contain; }
```

§ 10.4.2. Images

User agents are expected to render `<img>` elements and `<input>` elements whose type attributes are in the [image button](#) state, according to the first applicable rules from the following list:

↪ If the element represents an image

The user agent is expected to treat the element as a [replaced element](#) and render the image according to the rules for doing so defined in CSS.



If the element does not [represent](#) an image, but the element already has [intrinsic dimensions](#) (e.g., from the [dimension attributes](#) or CSS rules), and either:

- the user agent has reason to believe that the image will become *available* and be rendered in due course, or
- the element has no `alt` attribute, or
- the Document is in [quirks mode](#)

The user agent is expected to treat the element as a [replaced element](#) whose content is the text that the element represents, if any, optionally alongside an icon indicating that the image is being obtained (if applicable). For `<input>` elements, the element is expected to appear button-like to indicate that the element is a [button](#).

↪ If the element is an `<img>` element that represents some text and the user agent does not expect this to change

The user agent is expected to treat the element as a non-replaced phrasing element whose content is the text, op-

tionally with an icon indicating that an image is missing, so that the user can request the image be displayed or investigate why it is not rendering. In non-graphical contexts, such an icon should be omitted.

↪ **If the element is an `<img>` element that [represents](#) nothing and the user agent does not expect this to change**

The user agent is expected to treat the element as an empty inline element. (In the absence of further styles, this will cause the element to essentially not be rendered.)

↪ **If the element is an `<input>` element that does not [represent](#) an image and the user agent does not expect this to change**

The user agent is expected to treat the element as a [replaced element](#) consisting of a button whose content is the element's alternative text. The [intrinsic dimensions](#) of the button are expected to be about one line in height and whatever width is necessary to render the text on one line.

The icons mentioned above are expected to be relatively small so as not to disrupt most text but be easily clickable. In a visual environment, for instance, icons could be 16 pixels by 16 pixels square, or 1em by 1em if the images are scalable. In an audio environment, the icon could be a short bleep. The icons are intended to indicate to the user that they can be used to get to whatever options the user agent provides for images, and, where appropriate, are expected to provide access to the context menu that would have come up if the user interacted with the actual image.



All animated images with the same [absolute URL](#) and the same image data are expected to be rendered synchronized to the same timeline as a group, with the timeline starting at the time of the least recent addition to the group.

NOTE:

In other words, when a second image with the same [absolute URL](#) and animated image data is [inserted into a document](#), it jumps to the point in the animation cycle that is currently being displayed by the first image.

When a user agent is to **restart the animation** for an `<img>` element showing an animated image, all animated images with the same [absolute URL](#) and the same image data in that `<img>` element's [node document](#) are expected to restart their animation from the beginning.



The following CSS rules are expected to apply when the Document is in [quirks mode](#):

```
@namespace url(http://www.w3.org/1999/xhtml);

img[align=left i] { margin-right: 3px; }
img[align=right i] { margin-left: 3px; }
```

§ 10.4.3. Attributes for embedded content and images

The following CSS rules are expected to apply as [presentational hints](#):

```

@namespace url(http://www.w3.org/1999/xhtml);

iframe[frameborder=0], iframe[frameborder=no i] { border: none; }

applet[align=left i], embed[align=left i], iframe[align=left i],
img[align=left i], input[type=image i][align=left i], object[align=left i] {
    float: left;
}

applet[align=right i], embed[align=right i], iframe[align=right i],
img[align=right i], input[type=image i][align=right i], object[align=right i] {
    float: right;
}

applet[align=top i], embed[align=top i], iframe[align=top i],
img[align=top i], input[type=image i][align=top i], object[align=top i] {
    vertical-align: top;
}

applet[align=baseline i], embed[align=baseline i], iframe[align=baseline i],
img[align=baseline i], input[type=image i][align=baseline i], object[align=baseline i] {
    vertical-align: baseline;
}

applet[align=texttop i], embed[align=texttop i], iframe[align=texttop i],
img[align=texttop i], input[type=image i][align=texttop i], object[align=texttop i] {
    vertical-align: text-top;
}

applet[align=absmiddle i], embed[align=absmiddle i], iframe[align=absmiddle i],
img[align=absmiddle i], input[type=image i][align=absmiddle i], object[align=absmiddle i],
applet[align=abscenter i], embed[align=abscenter i], iframe[align=abscenter i],
img[align=abscenter i], input[type=image i][align=abscenter i], object[align=abscenter i] {
    vertical-align: middle;
}

applet[align=bottom i], embed[align=bottom i], iframe[align=bottom i],
img[align=bottom i], input[type=image i][align=bottom i],
object[align=bottom i] {
    vertical-align: bottom;
}

```

When an `<applet>`, `<embed>`, `<iframe>`, `<img>`, or `<object>` element, or an `<input>` element whose type attribute is in the [image button](#) state, has an `align` attribute whose value is an [ASCII case-insensitive](#) match for the string "center" or the string "middle", the user agent is expected to act as if the element's `'vertical-align'` property was set to a value that aligns the vertical middle of the element with the parent element's baseline.

The `hspace` attribute of `<applet>`, `<embed>`, `<iframe>`, `<img>`, or `<object>` elements, and input elements with a type attribute in the [image button](#) state, [maps to the dimension properties](#) `'margin-left'` and `'margin-right'` on the element.

The `vspace` attribute of `<applet>`, `<embed>`, `<iframe>`, `<img>`, or `<object>` elements, and input elements with a type attribute in the [image button](#) state, [maps to the dimension properties](#) `'margin-top'` and `'margin-bottom'` on the element.

When an `<img>` element, `<object>` element, or `<input>` element with a type attribute in the [image button](#) state has a `border` attribute whose value, when parsed using the [rules for parsing non-negative integers](#), is found to be a number greater than zero, the user agent is expected to use the parsed value for eight [presentational hints](#): four setting the parsed value as a pixel length for the element's `'border-top-width'`, `'border-right-width'`, `'border-bottom-width'`, and `'border-left-width'` properties, and four setting the element's `'border-top-style'`, `'border-right-style'`, `'border-bottom-style'`, and `'border-left-style'` properties to the value `'solid'`.

The `width` and `height` attributes on `<applet>`, `<embed>`, `<iframe>`, `<img>`, `object` or `<video>` elements, and `<input>` elements with a type attribute in the [image button](#) state and that either represents an image or that the user expects will eventually

represent an image, map to the dimension properties `'width'` and `'height'` on the element respectively.

§ 10.4.4. Image maps

Shapes on an [image map](#) are expected to act, for the purpose of the CSS cascade, as elements independent of the original [<area>](#) element that happen to match the same style rules but inherit from the [](#) or [<object>](#) element.

For the purposes of the rendering, only the `'cursor'` property is expected to have any effect on the shape.

EXAMPLE 653

Thus, for example, if an [<area>](#) element has a [style](#) attribute that sets the `'cursor'` property to `'help'`, then when the user designates that shape, the cursor would change to a Help cursor.

EXAMPLE 654

Similarly, if an [<area>](#) element had a CSS rule that set its `'cursor'` property to `'inherit'` (or if no rule setting the `'cursor'` property matched the element at all), the shape's cursor would be inherited from the [img](#) or [<object>](#) element of the [image map](#), not from the parent of the [<area>](#) element.

§ 10.5. Bindings

§ 10.5.1. Introduction

Exactly how the bindings are implemented is not specified by this specification. User agents are encouraged to make their bindings set the `'appearance'` CSS property appropriately to achieve platform-native appearances for widgets, and are expected to implement any relevant animations, etc, that are appropriate for the platform. [\[CSS-UI-3\]](#)

§ 10.5.2. The [<button>](#) element

When the *button* binding applies to a [<button>](#) element, the element is expected to render as an `'inline-block'` box rendered as a button whose contents are the contents of the element.

When the [<button>](#) element's `type` attribute is in the [Menu](#) state, the user agent is expected to indicate that activating the element will display a menu, e.g., by displaying a down-pointing triangle after the button's label.

§ 10.5.3. The [<details>](#) element

When the *details* binding applies to a [<details>](#) element, the element is expected to render as a `'block'` box with its `'padding-left'` property set to "40px" for left-to-right elements ([LTR-specific](#)) and with its `'padding-right'` property set to "40px" for right-to-left elements. The element's shadow tree is expected to take the element's first child [<summary>](#) element, if any, and place it in a first `'block'` box container, and then take the element's remaining descendants, if any, and place them in a second `'block'` box container.

The first container is expected to contain at least one line box, and that line box is expected to contain a disclosure widget (typically a triangle), horizontally positioned within the left padding of the [<details>](#) element. That widget is expected to allow the user to request that the details be shown or hidden.

The second container is expected to have its `'overflow'` property set to `'hidden'`. When the [<details>](#) element does not have an [open](#) attribute, this second container is expected to be removed from the rendering.

§ 10.5.4. The [<input>](#) element as a text entry widget

When the *input-textfield* binding applies to an [<input>](#) element whose `type` attribute is in the [Text](#), [Search](#), [Telephone](#), [URL](#), or [E-mail](#) state, the element is expected to render as an `'inline-block'` box rendered as a text field.

When the *input-password* binding applies to an [<input>](#) element whose `type` attribute is in the [Password](#) state, the element is expected to render as an `'inline-block'` box rendered as a text field whose contents are obscured.

If these text fields provide a text selection, then, when the user changes the current selection in such a binding, the user agent is expected to [queue a task](#) to [fire a simple event](#) that bubbles named `select` at the element, using the [user interaction task source](#) as the task source.

If an `<input>` element whose `type` attribute is in one of the above states has a `size` attribute, and parsing that attribute's value using the [rules for parsing non-negative integers](#) doesn't generate an error, then the user agent is expected to use the attribute as a [presentational hint](#) for the `'width'` property on the element, with the value obtained from applying the [converting a character width to pixels](#) algorithm to the value of the attribute.

If an `<input>` element whose `type` attribute is in one of the above states does *not* have a `size` attribute, then the user agent is expected to act as if it had a user-agent-level style sheet rule setting the `'width'` property on the element to the value obtained from applying the [converting a character width to pixels](#) algorithm to the number 20.

The [converting a character width to pixels](#) algorithm returns $(size - 1) \times avg + max$, where `size` is the character width to convert, `avg` is the average character width of the primary font for the element for which the algorithm is being run, in pixels, and `max` is the maximum character width of that same font, also in pixels. (The element's `'letter-spacing'` property does not affect the result.)

When the `input-textfield` binding applies to an element, the `'line-height'` property, if it has a computed value equivalent to a value that is less than 1.0, must have a used value of 1.0.

§ 10.5.5. The `<input>` element as domain-specific widgets

When the `input-date` binding applies to an `<input>` element whose `type` attribute is in the [Date](#) state, the element is expected to render as an `'inline-block'` box depicting a Date control.

When the `input-month` binding applies to an `<input>` element whose `type` attribute is in the [Month](#) state, the element is expected to render as an `'inline-block'` box depicting a Month control.

When the `input-week` binding applies to an `<input>` element whose `type` attribute is in the [Week](#) state, the element is expected to render as an `'inline-block'` box depicting a Week control.

When the `input-time` binding applies to an `<input>` element whose `type` attribute is in the [Time](#) state, the element is expected to render as an `'inline-block'` box depicting a Time control.

When the `input-datetime-local` binding applies to an `<input>` element whose `type` attribute is in the [Local Date and Time](#) state, the element is expected to render as an `'inline-block'` box depicting a Local Date and Time control.

When the `input-number` binding applies to an `<input>` element whose `type` attribute is in the [Number](#) state, the element is expected to render as an `'inline-block'` box depicting a Number control.

These controls are all expected to be about one line high, and about as wide as necessary to show the widest possible value.

§ 10.5.6. The `<input>` element as a range control

When the `input-range` binding applies to an `<input>` element whose `type` attribute is in the [Range](#) state, the element is expected to render as an `'inline-block'` box depicting a slider control.

When the control is wider than it is tall (or square), the control is expected to be a horizontal slider, with the lowest value on the right if the `'direction'` property on this element has a computed value of `'rtl'`, and on the left otherwise. When the control is taller than it is wide, it is expected to be a vertical slider, with the lowest value on the bottom.

Predefined suggested values (provided by the `list` attribute) are expected to be shown as tick marks on the slider, which the slider can snap to.

User agents are expected to use the used value of the `'direction'` property on the element to determine the direction in which the slider operates. Typically, a left-to-right (`'ltr'`) horizontal control would have the lowest value on the left and the highest value on the right, and vice versa.

§ 10.5.7. The `<input>` element as a color well

When the *input-color* binding applies to an `<input>` element whose type attribute is in the [Color](#) state, the element is expected to render as an ‘inline-block’ box depicting a color well, which, when activated, provides the user with a color picker (e.g., a color wheel or color palette) from which the color can be changed.

Predefined suggested values (provided by the `list` attribute) are expected to be shown in the color picker interface, not on the color well itself.

§ 10.5.8. The `<input>` element as a checkbox and radio button widgets

When the *input-checkbox* binding applies to an `<input>` element whose type attribute is in the [Checkbox](#) state, the element is expected to render as an ‘inline-block’ box containing a single checkbox control, with no label.

When the *input-radio* binding applies to an `<input>` element whose type attribute is in the [Radio Button](#) state, the element is expected to render as an ‘inline-block’ box containing a single radio button control, with no label.

§ 10.5.9. The `<input>` element as a file upload control

When the *input-file* binding applies to an `<input>` element whose type attribute is in the [File Upload](#) state, the element is expected to render as an ‘inline-block’ box containing a span of text giving the file name(s) of the [selected files](#), if any, followed by a button that, when activated, provides the user with a file picker from which the selection can be changed.

§ 10.5.10. The `<input>` element as a button

When the *input-button* binding applies to an `<input>` element whose type attribute is in the [submit button](#), [Reset Button](#), or [Button](#) state, the element is expected to render as an ‘inline-block’ box rendered as a button, about one line high, containing the contents of the element’s value attribute, if any, or text derived from the element’s type attribute in a user-agent-defined (and probably locale-specific) fashion, if not.

§ 10.5.11. The `<marquee>` element

When the *marquee* binding applies to a `<marquee>` element, while the element is [turned on](#), the element is expected to render in an animated fashion according to its attributes as follows:

If the element’s [behavior](#) attribute is in the [scroll](#) state

Slide the contents of the element in the direction described by the [direction](#) attribute as defined below, such that it begins off the start side of the `<marquee>`, and ends flush with the inner end side.

EXAMPLE 655

For example, if the [direction](#) attribute is [left](#) (the default), then the contents would start such that their left edge are off the side of the right edge of the marquee’s content area, and the contents would then slide up to the point where the left edge of the contents are flush with the left inner edge of the marquee’s content area.

Once the animation has ended, the user agent is expected to [increment the marquee current loop index](#). If the element is still [turned on](#) after this, then the user agent is expected to restart the animation.

If the element’s [behavior](#) attribute is in the [slide](#) state

Slide the contents of the element in the direction described by the [direction](#) attribute as defined below, such that it begins off the start side of the `<marquee>`, and ends off the end side of the `<marquee>`.

EXAMPLE 656

For example, if the [direction](#) attribute is [left](#) (the default), then the contents would start such that their left edge are off the side of the right edge of the `<marquee>`’s content area, and the contents would then slide up to the point where the *right* edge of the contents are flush with the left inner edge of the `<marquee>`’s content area.

Once the animation has ended, the user agent is expected to [increment the marquee current loop index](#). If the element is still [turned on](#) after this, then the user agent is expected to restart the animation.

If the element’s [behavior](#) attribute is in the [alternate](#) state

When the [marquee current loop index](#) is even (or zero), slide the contents of the element in the direction described by the [direction](#) attribute as defined below, such that it begins flush with the start side of the [<marquee>](#), and ends flush with the end side of the [<marquee>](#).

When the [marquee current loop index](#) is odd, slide the contents of the element in the opposite direction than that described by the [direction](#) attribute as defined below, such that it begins flush with the end side of the [<marquee>](#), and ends flush with the start side of the [<marquee>](#).

EXAMPLE 657

For example, if the [direction](#) attribute is [left](#) (the default), then the contents would with their right edge flush with the right inner edge of the marquee's content area, and the contents would then slide up to the point where the *left* edge of the contents are flush with the left inner edge of the marquee's content area.

Once the animation has ended, the user agent is expected to [increment the marquee current loop index](#). If the element is still [turned on](#) after this, then the user agent is expected to continue the animation.

The [direction](#) attribute has the meanings described in the following table:

direction attribute state	Direction of animation	Start edge	End edge	Opposite direction
left	← Right to left	Right	Left	→ Left to Right
right	→ Left to Right	Left	Right	← Right to left
up	↑ Up (Bottom to Top)	Bottom	Top	↓ Down (Top to Bottom)
down	↓ Down (Top to Bottom)	Top	Bottom	↑ Up (Bottom to Top)

In any case, the animation should proceed such that there is a delay given by the [marquee scroll interval](#) between each frame, and such that the content moves at most the distance given by the [marquee scroll distance](#) with each frame.

When a [<marquee>](#) element has a [bgcolor](#) attribute set, the value is expected to be parsed using the [rules for parsing a legacy color value](#), and if that does not return an error, the user agent is expected to treat the attribute as a [presentational hint](#) setting the element's ['background-color'](#) property to the resulting color.

The width and height attributes on a [<marquee>](#) element map to the dimension properties ['width'](#) and ['height'](#) on the element respectively.

The [intrinsic height](#) of a [<marquee>](#) element with its [direction](#) attribute in the [up](#) or [down](#) states is 200 CSS pixels.

The [vspace](#) attribute of a [<marquee>](#) element [maps to the dimension properties 'margin-top' and 'margin-bottom'](#) on the element. The [hspace](#) attribute of a [<marquee>](#) element [maps to the dimension properties 'margin-left' and 'margin-right'](#) on the element.

The ['overflow'](#) property on the [<marquee>](#) element is expected to be ignored; overflow is expected to always be hidden.

§ 10.5.12. The [<meter>](#) element

When the *meter* binding applies to a [<meter>](#) element, the element is expected to render as an ['inline-block'](#) box with a ['height'](#) of "1em" and a ['width'](#) of "5em", a ['vertical-align'](#) of "-0.2em", and with its contents depicting a gauge.

When the element is wider than it is tall (or square), the depiction is expected to be of a horizontal gauge, with the minimum value on the right if the ['direction'](#) property on this element has a computed value of ['rtl'](#), and on the left otherwise. When the element is taller than it is wide, it is expected to depict a vertical gauge, with the minimum value on the bottom.

User agents are expected to use a presentation consistent with platform conventions for gauges, if any.

NOTE:

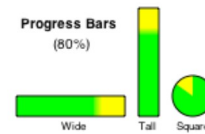
Requirements for what must be depicted in the gauge are included in the definition of the [<meter>](#) element.

§ 10.5.13. The [<progress>](#) element

When the *progress* binding applies to a [<progress>](#) element, the element is expected to render as an ['inline-block'](#) box with a

`'height'` of "1em" and a `'width'` of "10em", and a `'vertical-align'` of "-0.2em".

When the element is wider than it is tall, the element is expected to be depicted as a horizontal progress bar, with the start on the right and the end on the left if the `'direction'` property on this element has a computed value of `'rtl'`, and with the start on the left and the end on the right otherwise. When the element is taller than it is wide, it is expected to be depicted as a vertical progress bar, with the lowest value on the bottom. When the element is square, it is expected to be depicted as a direction-independent progress widget (e.g., a circular progress ring).



User agents are expected to use a presentation consistent with platform conventions for progress bars. In particular, user agents are expected to use different presentations for determinate and indeterminate progress bars. User agents are also expected to vary the presentation based on the dimensions of the element.

EXAMPLE 658

For example, on some platforms for showing indeterminate progress there is a "spinner" progress indicator with square dimensions, which could be used when the element is square, and an indeterminate progress bar, which could be used when the element is wide.

NOTE:

Requirements for how to determine if the progress bar is determinate or indeterminate, and what progress a determinate progress bar is to show, are included in the definition of the `<progress>` element.

§ 10.5.14. The `<select>` element

When the *select* binding applies to a `<select>` element whose `multiple` attribute is present, the element is expected to render as a multi-select list box.

When the *select* binding applies to a `<select>` element whose `multiple` attribute is absent, and the element's `display size` is greater than 1, the element is expected to render as a single-select list box.

When the element renders as a list box, it is expected to render as an `'inline-block'` box whose `'height'` is the height necessary to contain as many rows for items as given by the element's `display size`, or four rows if the attribute is absent, and whose `'width'` is the *width of the select's labels* plus the width of a scrollbar.

When the *select* binding applies to a `<select>` element whose `multiple` attribute is absent, and the element's `display size` is 1, the element is expected to render as a one-line drop down box whose width is the *width of the select's labels*.

In either case (list box or drop-down box), the element's items are expected to be the element's *list of options*, with the element's `<optgroup>` element children providing headers for groups of options where applicable.

An `<optgroup>` element is expected to be rendered by displaying the element's `label` attribute.

An `<option>` element is expected to be rendered by displaying the element's `label`, indented under its `<optgroup>` element if it has one.

The **width of the select's labels** is the wider of the width necessary to render the widest `<optgroup>`, and the width necessary to render the widest `<option>` element in the element's *list of options* (including its indent, if any).

If a `<select>` element contains a *placeholder label option*, the user agent is expected to render that option in a manner that conveys that it is a label, rather than a valid option of the control. This can include preventing the *placeholder label option* from being explicitly selected by the user. When the *placeholder label option*'s *selectedness* is true, the control is expected to be displayed in a fashion that indicates that no valid option is currently selected.

User agents are expected to render the labels in a select in such a manner that any alignment remains consistent whether the label is being displayed as part of the page or in a menu control.

§ 10.5.15. The `<textarea>` element

```
@namespace url(http://www.w3.org/1999/xhtml);
```

```
textarea { white-space: pre-wrap; }
```

When the *textarea* binding applies to a `<textarea>` element, the element is expected to render as an ‘inline-block’ box rendered as a multiline text field. If this text field provides a selection, then, when the user changes the current selection in such a binding, the user agent is expected to [queue a task](#) to [fire a simple event](#) that bubbles named `select` at the element, using the [user interaction task source](#) as the task source.

If the element has a `cols` attribute, and parsing that attribute’s value using the [rules for parsing non-negative integers](#) doesn’t generate an error, then the user agent is expected to use the attribute as a [presentational hint](#) for the ‘width’ property on the element, with the value being the [textarea effective width](#) (as defined below). Otherwise, the user agent is expected to act as if it had a user-agent-level style sheet rule setting the ‘width’ property on the element to the [textarea effective width](#).

The **textarea effective width** of a `<textarea>` element is $size \times avg + sbw$, where *size* is the element’s [character width](#), *avg* is the average character width of the primary font of the element, in CSS pixels, and *sbw* is the width of a scroll bar, in CSS pixels. (The element’s ‘letter-spacing’ property does not affect the result.)

If the element has a `rows` attribute, and parsing that attribute’s value using the [rules for parsing non-negative integers](#) doesn’t generate an error, then the user agent is expected to use the attribute as a [presentational hint](#) for the ‘height’ property on the element, with the value being the [textarea effective height](#) (as defined below). Otherwise, the user agent is expected to act as if it had a user-agent-level style sheet rule setting the ‘height’ property on the element to the [textarea effective height](#).

The **textarea effective height** of a `<textarea>` element is the height in CSS pixels of the number of lines specified the element’s [character height](#), plus the height of a scrollbar in CSS pixels.

User agents are expected to apply the ‘white-space’ CSS property to `<textarea>` elements. For historical reasons, if the element has a `wrap` attribute whose value is an [ASCII case-insensitive](#) match for the string “off”, then the user agent is expected to treat the attribute as a [presentational hint](#) setting the element’s ‘white-space’ property to ‘pre’.

§ 10.5.16. The `<keygen>` element

When the *keygen* binding applies to a `<keygen>` element, the element is expected to render as an ‘inline-block’ box containing a user interface to configure the key pair to be generated.

§ 10.6. Frames and framesets

User agent are expected to render `<frameset>` elements as a box with the height and width of the [viewport](#), with a surface rendered according to the following layout algorithm:

1. The `cols` and `rows` variables are lists of zero or more pairs consisting of a number and a unit, the unit being one of *percentage*, *relative*, and *absolute*.

Use the [rules for parsing a list of dimensions](#) to parse the value of the element’s `cols` attribute, if there is one. Let *cols* be the result, or an empty list if there is no such attribute.

Use the [rules for parsing a list of dimensions](#) to parse the value of the element’s `rows` attribute, if there is one. Let *rows* be the result, or an empty list if there is no such attribute.

2. For any of the entries in *cols* or *rows* that have the number zero and the unit *relative*, change the entry’s number to one.

3. If *cols* has no entries, then add a single entry consisting of the value 1 and the unit *relative* to *cols*.

If *rows* has no entries, then add a single entry consisting of the value 1 and the unit *relative* to *rows*.

4. Invoke the algorithm defined below to [convert a list of dimensions to a list of pixel values](#) using *cols* as the input list, and the width of the surface that the frameset is being rendered into, in CSS pixels, as the input dimension. Let *sized cols* be the resulting list.

Invoke the algorithm defined below to [convert a list of dimensions to a list of pixel values](#) using `rows` as the input list, and the height of the surface that the `frameset` is being rendered into, in CSS pixels, as the input dimension. Let `sized rows` be the resulting list.

5. Split the surface into a grid of $w \times h$ rectangles, where w is the number of entries in `sized cols` and h is the number of entries in `sized rows`.

Size the columns so that each column in the grid is as many CSS pixels wide as the corresponding entry in the `sized cols` list.

Size the rows so that each row in the grid is as many CSS pixels high as the corresponding entry in the `sized rows` list.

6. Let `children` be the list of frame and `<frameset>` elements that are children of the `<frameset>` element for which the algorithm was invoked.

7. For each row of the grid of rectangles created in the previous step, from top to bottom, run these substeps:

1. For each rectangle in the row, from left to right, run these substeps:

1. If there are any elements left in `children`, take the first element in the list, and assign it to the rectangle.

If this is a `<frameset>` element, then recurse the entire `frameset` layout algorithm for that `<frameset>` element, with the rectangle as the surface.

Otherwise, it is a `<frame>` element; render its [nested browsing context](#), positioned and sized to fit the rectangle.

2. If there are any elements left in `children`, remove the first element from `children`.

8. If the `<frameset>` element [has a border](#), draw an outer set of borders around the rectangles, using the element's [frame border color](#).

For each rectangle, if there is an element assigned to that rectangle, and that element [has a border](#), draw an inner set of borders around that rectangle, using the element's [frame border color](#).

For each (visible) border that does not about a rectangle that is assigned a `<frame>` element with a `noresize` attribute (including rectangles in further nested `<frameset>` elements), the user agent is expected to allow the user to move the border, resizing the rectangles within, keeping the proportions of any nested `frameset` grids.

A `frameset` or `<frame>` element **has a border** if the following algorithm returns true:

1. If the element has a `frameborder` attribute whose value is not the empty string and whose first character is either a U+0031 DIGIT ONE (1) character, a U+0079 LATIN SMALL LETTER Y character (y), or a U+0059 LATIN CAPITAL LETTER Y character (Y), then return true.
2. Otherwise, if the element has a `frameborder` attribute, return false.
3. Otherwise, if the element has a parent element that is a `<frameset>` element, then return true if *that* element [has a border](#), and false if it does not.
4. Otherwise, return true.

The **frame border color** of a `frameset` or `<frame>` element is the color obtained from the following algorithm:

1. If the element has a `bordercolor` attribute, and applying the [rules for parsing a legacy color value](#) to that attribute's value does not result in an error, then return the color so obtained.
2. Otherwise, if the element has a parent element that is a `<frameset>` element, then return the [frame border color](#) of that element.
3. Otherwise, return gray.

The algorithm to **convert a list of dimensions to a list of pixel values** consists of the following steps:

1. Let `input list` be the list of numbers and units passed to the algorithm.

Let *output list* be a list of numbers the same length as *input list*, all zero.

Entries in *output list* correspond to the entries in *input list* that have the same position.

2. Let *input dimension* be the size passed to the algorithm.

3. Let *count percentage* be the number of entries in *input list* whose unit is *percentage*.

Let *total percentage* be the sum of all the numbers in *input list* whose unit is *percentage*.

Let *count relative* be the number of entries in *input list* whose unit is *relative*.

Let *total relative* be the sum of all the numbers in *input list* whose unit is *relative*.

Let *count absolute* be the number of entries in *input list* whose unit is *absolute*.

Let *total absolute* be the sum of all the numbers in *input list* whose unit is *absolute*.

Let *remaining space* be the value of *input dimension*.

4. If *total absolute* is greater than *remaining space*, then for each entry in *input list* whose unit is *absolute*, set the corresponding value in *output list* to the number of the entry in *input list* multiplied by *remaining space* and divided by *total absolute*. Then, set *remaining space* to zero.

Otherwise, for each entry in *input list* whose unit is *absolute*, set the corresponding value in *output list* to the number of the entry in *input list*. Then, decrement *remaining space* by *total absolute*.

5. If *total percentage* multiplied by the *input dimension* and divided by 100 is greater than *remaining space*, then for each entry in *input list* whose unit is *percentage*, set the corresponding value in *output list* to the number of the entry in *input list* multiplied by *remaining space* and divided by *total percentage*. Then, set *remaining space* to zero.

Otherwise, for each entry in *input list* whose unit is *percentage*, set the corresponding value in *output list* to the number of the entry in *input list* multiplied by the *input dimension* and divided by 100. Then, decrement *remaining space* by *total percentage* multiplied by the *input dimension* and divided by 100.

6. For each entry in *input list* whose unit is *relative*, set the corresponding value in *output list* to the number of the entry in *input list* multiplied by *remaining space* and divided by *total relative*.

7. Return *output list*.

User agents working with integer values for frame widths (as opposed to user agents that can lay frames out with subpixel accuracy) are expected to distribute the remainder first to the last entry whose unit is *relative*, then equally (not proportionally) to each entry whose unit is *percentage*, then equally (not proportionally) to each entry whose unit is *absolute*, and finally, failing all else, to the last entry.



The contents of a `<frame>` element that does not have a `frameset` parent are expected to be rendered as transparent black; the user agent is expected to not render the [nested browsing context](#) in this case, and that [nested browsing context](#) is expected to have a [viewport](#) with zero width and zero height.

§ 10.7. Interactive media

§ 10.7.1. Links, forms, and navigation

User agents are expected to allow the user to control aspects of [hyperlink](#) activation and [§4.10.22 Form submission](#), such as which [browsing context](#) is to be used for the subsequent [navigation](#).

User agents are expected to allow users to discover the destination of [hyperlinks](#) and of [forms](#) before triggering their [navigation](#).

User agents may allow users to [navigate browsing contexts](#) to the URLs indicated by the `<cite>` attributes on `<q>`, `<blockquote>`, `<ins>`, and `<del>` elements.

User agents may surface [hyperlinks](#) created by link elements in their user interface.

NOTE:

While `<link>` elements that create [hyperlinks](#) will match the `:link` or `:visited` pseudo-classes, will react to clicks if visible, and so forth, this does not extend to any browser interface constructs that expose those same links. Activating a link through the browser's interface, rather than in the page itself, does not trigger click events and the like.

§ 10.7.2. The `title` attribute

User agents are expected to expose the [advisory information](#) of elements upon user request, and to make the user aware of the presence of such information.

On interactive graphical systems where the user can use a pointing device, this could take the form of a tooltip. When the user is unable to use a pointing device, then the user agent is expected to make the content available in some other fashion, e.g., by making the element a *focusable area* and always displaying the [advisory information](#) of the currently [focused](#) element.

U+000A LINE FEED (LF) characters are expected to cause line breaks in the tooltip; U+0009 CHARACTER TABULATION (tab) characters are expected to render as a non-zero horizontal shift that lines up the next glyph with the next tab stop, with tab stops occurring at points that are multiples of 8 times the width of a U+0020 SPACE character.

EXAMPLE 659

For example, a visual user agent could make elements with a `title` attribute [focusable](#), and could make any [focused](#) element with a `title` attribute show its tooltip under the element while the element has focus. This would allow a user to tab around the document to find all the advisory text.

EXAMPLE 660

As another example, a screen reader could provide an audio cue when reading an element with a tooltip, with an associated key to read the last tooltip for which a cue was played.

§ 10.7.3. Editing hosts

The current text editing caret (i.e., the [active range](#), if it is empty and in an [editing host](#)), if any, is expected to act like an inline [replaced element](#) with the vertical dimensions of the caret and with zero width for the purposes of the CSS rendering model.

NOTE:

This means that even an empty block can have the caret inside it, and that when the caret is in such an element, it prevents margins from collapsing through the element.

§ 10.7.4. Text rendered in native user interfaces

User agents are expected to honor the Unicode semantics of text that is exposed in user interfaces, for example supporting the bidirectional algorithm in text shown in dialogs, title bars, pop-up menus, and tooltips. Text from the contents of elements is expected to be rendered in a manner that honors [the directionality](#) of the element from which the text was obtained. Text from attributes is expected to be rendered in a manner that honours the [directionality of the attribute](#).

EXAMPLE 661

Consider the following markup, which has Hebrew text asking for a programming language, the languages being text for which a left-to-right direction is important given the punctuation in some of their names:

```
<p dir="rtl" lang="he">
<label>
בחר שפת תכנות:
<select>
  <option dir="ltr">C++</option>
  <option dir="ltr">C#</option>
  <option dir="ltr">FreePascal</option>
  <option dir="ltr">F#</option>
</select>
</label>
</p>
```

If the `<select>` element was rendered as a drop down box, a correct rendering would ensure that the punctuation was the same both in the drop down, and in the box showing the current selection.



EXAMPLE 662

The directionality of attributes depends on the attribute and on the element's `dir` attribute, as the following example demonstrates. Consider this markup:

```
<table>
<tr>
<th abbr="(A" dir="ltr">A
<th abbr=")A" dir="rtl">A
<th abbr="(A" dir="auto">A
</table>
```

If the `abbr` attributes are rendered, e.g., in a tooltip or other user interface, the first will have a left parenthesis (because the direction is `'ltr'`), the second will have a right parenthesis (because the direction is `'rtl'`), and the third will have a right parenthesis (because the direction is determined *from the attribute value* to be `'rtl'`).

However, if instead the attribute was not a [directionality-capable attribute](#), the results would be different:

```
<table>
<tr>
<th>A
<th>A
<th>A
</table>
```

In this case, if the user agent were to expose the `data-abbrev` attribute in the user interface (e.g., in a debugging environment), the last case would be rendered with a *left* parenthesis, because the direction would be determined from the element's contents.

A string provided by a script (e.g., the argument to `window.alert()`) is expected to be treated as an independent set of one or more bidirectional algorithm paragraphs when displayed, as defined by the bidirectional algorithm, including, for instance, supporting the paragraph-breaking behavior of U+000A LINE FEED (LF) characters. For the purposes of determining the paragraph level of such text in the bidirectional algorithm, this specification does *not* provide a higher-level override of rules P2 and P3. [\[BIDI\]](#)

When necessary, authors can enforce a particular direction for a given paragraph by starting it with the Unicode U+200E LEFT-TO-RIGHT MARK or U+200F RIGHT-TO-LEFT MARK characters.

EXAMPLE 663

Thus, the following script:

```
alert('\u05DC\u05DE\u05D3 HTML \u05D4\u05D9\u05D5\u05DD!')
```

...would always result in a message reading "למד HTML היום!" (not "היום HTML למד!"), regardless of the language of the user agent interface or the direction of the page or any of its elements.

EXAMPLE 664

For a more complex example, consider the following script:

```
/* Warning: this script does not handle right-to-left scripts correctly */
var s;
if (s = prompt('What is your name?')) {
  alert(s + '! Ok, Fred, ' + s + ', and Wilma will get the car.');
```

When the user enters "Kitty", the user agent would alert "Kitty! Ok, Fred, Kitty, and Wilma will get the car.". However, if the user enters "لا أفهم", then the bidirectional algorithm will determine that the direction of the paragraph is right-to-left, and so the output will be the following unintended mess:

"لا أفهم! Ok, Fred, لا أفهم, and Wilma will get the car"

To force an alert that starts with user-provided text (or other text of unknown directionality) to render left-to-right, the string can be prefixed with a U+200E LEFT-TO-RIGHT MARK character:

```
var s;
if (s = prompt('What is your name?')) {
  alert('\u200E' + s + '! Ok, Fred, ' + s + ', and Wilma will get the car.');
```

§ 10.8. Print media

User agents are expected to allow the user to request the opportunity to **obtain a physical form** (or a representation of a physical form) of a [Document](#). For example, selecting the option to print a page or convert it to PDF format. [\[PDF\]](#)

When the user actually [obtains a physical form](#) (or a representation of a physical form) of a [Document](#), the user agent is expected to create a new rendering of the [Document](#) for the print media.

§ 10.9. Unstyled XML documents

HTML user agents may, in certain circumstances, find themselves rendering non-HTML documents that use vocabularies for which they lack any built-in knowledge. This section provides for a way for user agents to handle such documents in a somewhat useful manner.

While a [Document](#) is an [unstyled document](#), the user agent is expected to render [an unstyled document view](#).

A [Document](#) is an **unstyled document** while it matches the following conditions:

- The [Document](#) has no author style sheets (whether referenced by HTTP headers, processing instructions, elements like `<link>`, inline elements like `<style>`, or any other mechanism).
- None of the elements in the [Document](#) have any [presentational hints](#).
- None of the elements in the [Document](#) have any [CSS styling attributes](#).

- None of the elements in the [Document](#) are in any of the following namespaces: [HTML namespace](#), [SVG namespace](#), [MathML namespace](#)
- The [Document](#) has no *focusable area* (e.g., from XLink) other than the [viewport](#).
- The [Document](#) has no [hyperlinks](#) (e.g., from XLink).
- There exists no [script](#) whose [settings object](#) specifies this [Document](#) as the [responsible document](#).
- None of the elements in the [Document](#) have any registered event listeners.

An **unstyled document view** is one where the DOM is not rendered according to CSS (which would, since there are no applicable styles in this context, just result in a wall of text), but is instead rendered in a manner that is useful for a developer. This could consist of just showing the [Document](#) object's source, maybe with syntax highlighting, or it could consist of displaying just the DOM tree, or simply a message saying that the page is not a styled document.

NOTE:

If a [Document](#) stops being an [unstyled document](#), then the conditions above stop applying, and thus a user agent following these requirements will switch to using the regular CSS rendering.

§ 11. Obsolete features

§ 11.1. Obsolete but conforming features

Features listed in this section will trigger warnings in conformance checkers.

Authors should not specify a [border](#) attribute on an [](#) element. If the attribute is present, its value must be the string "0". CSS should be used instead.

Authors should not specify a [language](#) attribute on a [<script>](#) element. If the attribute is present, its value must be an [ASCII case-insensitive](#) match for the string "JavaScript" and either the [type](#) attribute must be omitted or its value must be an [ASCII case-insensitive](#) match for the string "text/javascript". The attribute should be entirely omitted instead (with the value "JavaScript", it has no effect), or replaced with use of the [type](#) attribute.

Authors should not specify the [name](#) attribute on [<a>](#) elements. If the attribute is present, its value must not be the empty string and must neither be equal to the value of any of the [IDs](#) in the element's [home subtree](#) other than the element's own [id](#), if any, nor be equal to the value of any of the other [name](#) attributes on [<a>](#) elements in the element's [home subtree](#). If this attribute is present and the element has an [id](#), then the attribute's value must be equal to the element's [id](#). In earlier versions of the language, this attribute was intended as a way to specify possible targets for fragment identifiers in [URLs](#). The [id](#) attribute should be used instead.

Authors should not, but may despite requirements to the contrary elsewhere in this specification, specify the [maxlength](#) and [size](#) attributes on [<input>](#) elements whose [type](#) attributes are in the [Number](#) state. One valid reason for using these attributes regardless is to help legacy user agents that do not support [<input>](#) elements with `type="number"` to still render the text field with a useful width.

NOTE:

In [the HTML syntax](#), specifying a [DOCTYPE](#) that is an [obsolete permitted DOCTYPE](#) will also trigger a warning.

§ 11.1.1. Warnings for obsolete but conforming features

To ease the transition from HTML Transitional documents to the language defined in *this* specification, and to discourage certain features that are only allowed in very few circumstances, conformance checkers must warn the user when the following features are used in a document. These are generally old obsolete features that have no effect, and are allowed only to distinguish between likely mistakes (regular conformance errors) and mere vestigial markup or unusual and discouraged practices (these warnings).

The following features must be categorized as described above:

- The presence of an [obsolete permitted DOCTYPE](#) in an [HTML document](#).
- The presence of a [border](#) attribute on an [](#) element if its value is the string "0".
- The presence of a [language](#) attribute on a [<script>](#) element if its value is an [ASCII case-insensitive](#) match for the string "JavaScript" and if there is no [type](#) attribute or there is and its value is an [ASCII case-insensitive](#) match for the string "text/javascript".
- The presence of a [name](#) attribute on an [<a>](#) element, if its value is not the empty string.
- The presence of a [maxlength](#) attribute on an [<input>](#) element whose [type](#) attribute is in the [Number](#) state.
- The presence of a [size](#) attribute on an [<input>](#) element whose [type](#) attribute is in the [Number](#) state.

Conformance checkers must distinguish between pages that have no conformance errors and have none of these obsolete features, and pages that have no conformance errors but do have some of these obsolete features.

EXAMPLE 665

For example, a validator could report some pages as "Valid HTML" and others as "Valid HTML with warnings".

§ 11.2. Non-conforming features

Elements in the following list are entirely obsolete, and must not be used by authors:

applet

Use [<embed>](#) or [<object>](#) instead.

acronym

Use [<abbr>](#) instead.

bgsound

Use [<audio>](#) instead.

dir

Use [](#) instead.

frame

frameset

noframes

Either use [<iframe>](#) and CSS instead, or use server-side includes to generate complete pages with the various invariant parts merged in.

isindex

Use an explicit [<form>](#) and [text field](#) combination instead.

listing

Use [<pre>](#) and [<code>](#) instead.

nextid

Use GUIDs instead.

noembed

Use [<object>](#) instead of [<embed>](#) when fallback is necessary.

plaintext

Use the "text/plain" [MIME type](#) instead.

strike

Use `del` instead if the element is marking an edit, otherwise use `s` instead.

xmp

Use [<pre>](#) and [<code>](#) instead, and escape "<" and "&" characters as "<" and "&" respectively.

basefont

big

*blink**center**font*[<marquee>](#)*multicol**nobr**spacer**tt*

Use appropriate elements or CSS instead.

Where the [<tt>](#) element would have been used for marking up keyboard input, consider the [<kbd>](#) element; for variables, consider the [<var>](#) element; for computer code, consider the [<code>](#) element; and for computer output, consider the [<samp>](#) element.

Similarly, if the [<big>](#) element is being used to denote a heading, consider using the `h1` element; if it is being used for marking up important passages, consider the [](#) element; and if it is being used for highlighting text for reference purposes, consider the [<mark>](#) element.

See also the [text-level semantics usage summary](#) for more suggestions with examples.



The following attributes are obsolete (though the elements are still part of the language), and must not be used by authors:

charset on [<a>](#) elements

charset on [<link>](#) elements

Use an HTTP Content-Type header on the linked resource instead.

coords on [<a>](#) elements

shape on [<a>](#) elements

Use `area` instead of [<a>](#) for image maps.

methods on [<a>](#) elements

methods on [<link>](#) elements

Use the HTTP OPTIONS feature instead.

name on [<a>](#) elements (except as noted in the previous section)

name on [<embed>](#) elements

name on [](#) elements

name on [<option>](#) elements

Use the `id` attribute instead.

urn on [<a>](#) elements

urn on [<link>](#) elements

Specify the preferred persistent identifier using the [href](#) attribute instead.

accept on [<form>](#) elements

Use the [accept](#) attribute directly on the [<input>](#) elements instead.

type on [<area>](#) elements

These attributes do not do anything useful, and for historical reasons there are no corresponding IDL attributes on [<area>](#) elements. Omit them altogether.

nohref on [<area>](#) elements

Omitting the [href](#) attribute is sufficient; the `nohref` attribute is unnecessary. Omit it altogether.

profile on [<head>](#) elements

When used for declaring which meta terms are used in the document, unnecessary; omit it altogether, and [register the names](#).

When used for triggering specific user agent behaviors: use a [<link>](#) element instead.

version on <html> elements

Unnecessary. Omit it altogether.

manifest on <html> elements

The use of application caches is not recommended. Alternative mechanisms to support offline applications include the use of [\[WEBSTORAGE\]](#), [\[IndexedDB\]](#), and [\[SERVICE-WORKERS\]](#).

ismap on <input> elements

Unnecessary. Omit it altogether. All [<input>](#) elements with a type attribute in the [image button](#) state are processed as server-side image maps.

usemap on <input> elements

Use `img` instead of `input` for image maps.

lowsrc on elements

Use a progressive JPEG image (given in the `src` attribute), instead of using two separate images.

target on <link> elements

Unnecessary. Omit it altogether.

scheme on <meta> elements

Use only one scheme per field, or make the scheme declaration part of the value.

archive on <object> elements

classid on <object> elements

code on <object> elements

codebase on <object> elements

codetype on <object> elements

Use the [data](#) and [type](#) attributes to invoke [plugins](#). To set parameters with these names in particular, the [<param>](#) element can be used.

declare on <object> elements

Repeat the [<object>](#) element completely each time the resource is to be reused.

standby on <object> elements

Optimize the linked resource so that it loads quickly or, at least, incrementally.

usemap on <object> elements

Use `img` instead of `object` for image maps.

type on <param> elements

valuetype on <param> elements

Use the name and value attributes without declaring value types.

Language on <script> elements (except as noted in the previous section)

Use the type attribute instead.

event on <script> elements

for on <script> elements

Use DOM events mechanisms to register event listeners. [\[DOM\]](#)

media on <source> elements

Use script to select the media resource(s) to use.

datapagesize on <table> elements

Unnecessary. Omit it altogether.

summary on <table> elements

Use one of the [§4.9.1.1 Techniques for describing tables](#) given in the `table` section instead.

abbr on <td> elements

Use text that begins in an unambiguous and terse manner, and include any more elaborate text after that. The `title` at-

tribute can also be useful in including more detailed text, so that the cell's contents can be made terse. If it's a heading, use `th` (which has an `abbr` attribute).

axis on `<td>` and `<th>` elements

Use the `scope` attribute on the relevant `<th>`.

scope on `<td>` elements

Use `<th>` elements for heading cells.

datasrc on `<a>`, `<applet>`, `<button>`, `<div>`, `<frame>`, `<iframe>`, `<img>`, `<input>`, `<label>`, `<legend>`, `<marquee>`, `<object>`, `<option>`, `<select>`, `<span>`, `<table>`, and `<textarea>` elements

datafld on `<a>`, `<applet>`, `<button>`, `<div>`, `<fieldset>`, `<frame>`, `<iframe>`, `<img>`, `<input>`, `<label>`, `<legend>`, `<marquee>`, `<object>`, `<param>`, `<select>`, `<span>`, and `<textarea>` elements

dataformatas on `<button>`, `<div>`, `<input>`, `<label>`, `<legend>`, `<marquee>`, `<object>`, `<option>`, `<select>`, `<span>`, and `<table>` elements

Use script and a mechanism such as XMLHttpRequest to populate the page dynamically. [XHR]

alink on `<body>` elements

bgcolor on `<body>` elements

bottommargin on `<body>` elements

leftmargin on `<body>` elements

link on `<body>` elements

marginheight on `<body>` elements

marginwidth on `<body>` elements

rightmargin on `<body>` elements

text on `<body>` elements

margin on `<body>` elements

vlink on `<body>` elements

clear on `<br>` elements

align on `<caption>` elements

align on `<col>` elements

char on `<col>` elements

charoff on `<col>` elements

valign on `<col>` elements

width on `<col>` elements

align on `<div>` elements

compact on `<dl>` elements

align on `<embed>` elements

hspace on `<embed>` elements

vspace on `<embed>` elements

align on `<hr>` elements

color on `<hr>` elements

noshade on `<hr>` elements

size on `<hr>` elements

width on `<hr>` elements

align on `<h1>`—`<h6>` elements

align on `<iframe>` elements

allowtransparency on `<iframe>` elements

frameborder on `<iframe>` elements

framespacing on `<iframe>` elements

hspace on `<iframe>` elements

marginheight on `<iframe>` elements

marginwidth on `<iframe>` elements

scrolling on `<iframe>` elements

vspace on `<iframe>` elements

align on [<input>](#) elements
border on [<input>](#) elements
hspace on [<input>](#) elements
vspace on [<input>](#) elements
align on [](#) elements
border on [](#) elements (except as noted in the previous section)
hspace on [](#) elements
vspace on [](#) elements
align on [<legend>](#) elements
type on [](#) elements
compact on [<menu>](#) elements
align on [<object>](#) elements
border on [<object>](#) elements
hspace on [<object>](#) elements
vspace on [<object>](#) elements
compact on [](#) elements
align on [<p>](#) elements
width on [<pre>](#) elements
align on [<table>](#) elements
bgcolor on [<table>](#) elements
border on [<table>](#) elements
bordercolor on [<table>](#) elements
cellpadding on [<table>](#) elements
cellspacing on [<table>](#) elements
frame on [<table>](#) elements
height on [<table>](#) elements
rules on [<table>](#) elements
width on [<table>](#) elements
align on [<tbody>](#), [<thead>](#), and [<tfoot>](#) elements
char on [<tbody>](#), [<thead>](#), and [<tfoot>](#) elements
charoff on [<tbody>](#), [<thead>](#), and [<tfoot>](#) elements
valign on [<tbody>](#), [<thead>](#), and [<tfoot>](#) elements
align on [<td>](#) and [<th>](#) elements
bgcolor on [<td>](#) and [<th>](#) elements
char on [<td>](#) and [<th>](#) elements
charoff on [<td>](#) and [<th>](#) elements
height on [<td>](#) and [<th>](#) elements
nowrap on [<td>](#) and [<th>](#) elements
valign on [<td>](#) and [<th>](#) elements
width on [<td>](#) and [<th>](#) elements
align on [<tr>](#) elements
bgcolor on [<tr>](#) elements
char on [<tr>](#) elements
charoff on [<tr>](#) elements
height on [<tr>](#) elements
valign on [<tr>](#) elements
compact on [](#) elements
type on [](#) elements
background on [<body>](#), [<table>](#), [<thead>](#), [<tbody>](#), [<tfoot>](#), [<tr>](#), [<td>](#), and [<th>](#) elements

Use CSS instead.



The [border](#) attribute on the [<table>](#<table>) element can be used to provide basic fallback styling for the purpose of making tables legible in browsing environments where CSS support is limited or absent, such as text-based browsers, WYSIWYG editors, and in situations where CSS support is disabled or the style sheet is lost. Only the empty string and the value "1" may be used as [border](#) values for this purpose. Other values are considered obsolete. To regulate the thickness of such borders, authors should instead use CSS.

§ 11.3. Requirements for implementations

§ 11.3.1. The `<applet>` element

This feature is in the process of being removed from the Web platform. (This is a long process that takes many years.) Using the [<applet>](#<applet>) element at this time is highly discouraged.

The [<applet>](#<applet>) element is a Java-specific variant of the [<embed>](#<embed>) element. The [<applet>](#<applet>) element is now obsoleted so that all extension frameworks (Java, .NET, Flash, etc) are handled in a consistent manner.

When the element matches any of the following conditions, it [represents](#) its contents:

- The element is still in the [stack of open elements](#) of an [HTML parser](#) or [XML parser](#).
- The element is not [in a Document](#).
- The element's [node document](#) is not [fully active](#).
- The element's [node document](#)'s [active sandboxing flag set](#) has its [sandboxed plugins browsing context flag](#) set.
- The element has an ancestor [media element](#).
- The element has an ancestor [<object>](#<object>) element that is *not* showing its [fallback content](#).
- No Java Language runtime [plugin](#) is available.
- A Java runtime [plugin](#) is available but it is disabled.

Otherwise, the user agent should instantiate a Java Language runtime [plugin](#), and should pass the names and values of all the attributes on the element, in the order they were added to the element, with the attributes added by the parser being ordered in source order, and then a parameter named "PARAM" whose value is null, and then all the names and values of [parameters](#) given by [<param>](#<param>) elements that are children of the [<applet>](#<applet>) element, in [tree order](#), to the [plugin](#) used. If the [plugin](#) supports a scriptable interface, the `HTMLAppletElement` object representing the element should expose that interface. The [<applet>](#<applet>) element [represents](#) the [plugin](#).

NOTE:

The [<applet>](#<applet>) element is unaffected by the CSS `'display'` property. The Java Language runtime is instantiated even if the element is hidden with a `'display:none'` CSS style.

The [<applet>](#<applet>) element must implement the `HTMLAppletElement` interface.

```

interface HTMLAppletElement : HTMLElement {
  attribute DOMString align;
  attribute DOMString alt;
  attribute DOMString archive;
  attribute DOMString code;
  attribute DOMString codeBase;
  attribute DOMString height;
  attribute unsigned long hspace;
  attribute DOMString name;
  attribute DOMString _object; // the underscore is not part of the identifier
  attribute unsigned long vspace;
  attribute DOMString width;
};

```

The **align**, **alt**, **archive**, **code**, **height**, **hspace**, **name**, **object**, **vspace**, and **width** IDL attributes must [reflect](#) the respective content attributes of the same name. For the purposes of reflection, the `<applet>` element's **object** content attribute is defined as containing a [URL](#).

The **codeBase** IDL attribute must [reflect](#) the codebase content attribute, which for the purposes of reflection is defined as containing a [URL](#).

§ 11.3.2. The `marquee` element

The `<marquee>` element is a presentational element that animates content. CSS transitions and animations are a more appropriate mechanism. [\[CSS3-ANIMATIONS\]](#) [\[CSS3-TRANSITIONS\]](#)

The [task source](#) for tasks mentioned in this section is the [DOM manipulation task source](#).

The `<marquee>` element must implement the `HTMLMarqueeElement` interface.

```

interface HTMLMarqueeElement : HTMLElement {
  attribute DOMString behavior;
  attribute DOMString bgColor;
  attribute DOMString direction;
  attribute DOMString height;
  attribute unsigned long hspace;
  attribute long loop;
  attribute unsigned long scrollAmount;
  attribute unsigned long scrollDelay;
  attribute boolean trueSpeed;
  attribute unsigned long vspace;
  attribute DOMString width;

  attribute EventHandler onbounce;
  attribute EventHandler onfinish;
  attribute EventHandler onstart;

  void start();
  void stop();
};

```

A `<marquee>` element can be **turned on** or **turned off**. When it is created, it is [turned on](#).

When the `start()` method is called, the `<marquee>` element must be [turned on](#).

When the `stop()` method is called, the `<marquee>` element must be [turned off](#).

When a `<marquee>` element is created, the user agent must [queue a task](#) to [fire a simple event](#) named `start` at the element.



The **behavior** content attribute on `<marquee>` elements is an [enumerated attribute](#) with the following keywords (all non-conforming):

Keyword	State
scroll	scroll
slide	slide
alternate	alternate

The *missing value default* is the [scroll](#) state.



The **direction** content attribute on `<marquee>` elements is an [enumerated attribute](#) with the following keywords (all non-conforming):

Keyword	State
left	left
right	right
up	up
down	down

The *missing value default* is the [left](#) state.



The **truespeed** content attribute on `<marquee>` elements is a [boolean attribute](#).



A `<marquee>` element has a **marquee scroll interval**, which is obtained as follows:

1. If the element has a **scrolldelay** content attribute, and parsing its value using the [rules for parsing non-negative integers](#) does not return an error, then let *delay* be the parsed value. Otherwise, let *delay* be 85.
2. If the element does not have a [truespeed](#) attribute, and the *delay* value is less than 60, then let *delay* be 60 instead.
3. The **marquee scroll interval** is *delay*, interpreted in milliseconds.



A `<marquee>` element has a **marquee scroll distance**, which, if the element has a **scrollamount** content attribute, and parsing its value using the [rules for parsing non-negative integers](#) does not return an error, is the parsed value interpreted in CSS pixels, and otherwise is 6 CSS pixels.



A `<marquee>` element has a **marquee loop count**, which, if the element has a **loop** content attribute, and parsing its value using the [rules for parsing integers](#) does not return an error or a number less than 1, is the parsed value, and otherwise is -1.

The **loop** IDL attribute, on getting, must return the element's [marquee loop count](#); and on setting, if the new value is different than the element's [marquee loop count](#) and either greater than zero or equal to -1, must set the element's [loop](#) content attribute (adding it if necessary) to the [valid integer](#) that represents the new value. (Other values are ignored.)

A `<marquee>` element also has a **marquee current loop index**, which is zero when the element is created.

The rendering layer will occasionally **increment the marquee current loop index**, which must cause the following steps to

be run:

1. If the [marquee loop count](#) is -1, then abort these steps.
 2. Increment the [marquee current loop index](#) by one.
 3. If the [marquee current loop index](#) is now equal to or greater than the element's [marquee loop count](#), [turn off](#) the [<marquee>](#) element and [queue a task](#) to [fire a simple event](#) named [finish](#) at the [<marquee>](#) element.
- Otherwise, if the [behavior](#) attribute is in the [alternate](#) state, then [queue a task](#) to [fire a simple event](#) named [bounce](#) at the [<marquee>](#) element.
- Otherwise, [queue a task](#) to [fire a simple event](#) named [start](#) at the [<marquee>](#) element.



The following are the [event handlers](#) (and their corresponding [event handler event types](#)) that must be supported, as [event handler content attributes](#) and [event handler IDL attributes](#), by [<marquee>](#) elements:

Event handler	Event handler event type
onbounce	bounce
onfinish	finish
onstart	start



The **behavior**, **direction**, **height**, **hspace**, **vspace**, and **width** IDL attributes must [reflect](#) the respective content attributes of the same name.

The **bgColor** IDL attribute must [reflect](#) the **bgcolor** content attribute.

The **scrollAmount** IDL attribute must [reflect](#) the [scrollamount](#) content attribute. The default value is 6.

The **scrollDelay** IDL attribute must [reflect](#) the [scrolldelay](#) content attribute. The default value is 85.

The **trueSpeed** IDL attribute must [reflect](#) the [truespeed](#) content attribute.

§ 11.3.3. Frames

The **frameset** element acts as the [<body>](#) element in documents that use frames.

The [<frameset>](#) element must implement the `HTMLFrameSetElement` interface.

```
interface HTMLFrameSetElement : HTMLElement {
    attribute DOMString cols;
    attribute DOMString rows;
};
HTMLFrameSetElement implements WindowEventHandlers;
```

The **cols** and **rows** IDL attributes of the [<frameset>](#) element must [reflect](#) the respective content attributes of the same name.

The [<frameset>](#) element exposes as [event handler content attributes](#) a number of the [event handlers](#) of the `Window` object. It also mirrors their [event handler IDL attributes](#).

The [onblur](#), [onerror](#), [onfocus](#), [onload](#), [onresize](#), and [onscroll](#) [event handlers](#) of the `Window` object, exposed on the [<frameset>](#) element, replace the generic [event handlers](#) with the same names normally supported by [html elements](#).



The **frame** element defines a [nested browsing context](#) similar to the `<iframe>` element, but rendered within a `<frameset>` element.

A `<frame>` element is said to be an **active `<frame>` element** when it is [in a Document](#).

When a `<frame>` element is created as an [active frame element](#), or becomes an [active frame element](#) after not having been one, the user agent must create a [nested browsing context](#), and then [process the frame attributes](#) for the first time.

When a `<frame>` element stops being an [active frame element](#), the user agent must [discard](#) the [nested browsing context](#).

Whenever a `<frame>` element with a [nested browsing context](#) has its `src` attribute set, changed, or removed, the user agent must [process the frame attributes](#).

When the user agent is to **process the `<frame>` attributes**, it must run the first appropriate steps from the following list:

- If the element has no `src` attribute specified, and the user agent is processing the `<frame>`'s attributes for the first time

[Queue a task](#) to [fire a simple event](#) named `load` at the `<frame>` element.

- Otherwise

1. If the value of the `src` attribute is the empty string, let `url` be the string `"about:blank"`.

Otherwise, [resolve](#) the value of the `src` attribute, relative to the `<frame>` element.

If that is not successful, then let `url` be the string `"about:blank"`. Otherwise, let `url` be the resulting [absolute URL](#).

2. [Navigate](#) the element's [child browsing context](#) to `url`.

Furthermore, if the [active document](#) of the element's [child browsing context](#) before such a [navigation](#) was not [completely loaded](#) at the time of the new [navigation](#), then the [navigation](#) must be completed with [replacement enabled](#).

Similarly, if the [child browsing context](#)'s [session history](#) contained only one [Document](#) when the [process the frame attributes](#) algorithm was invoked, and that was the `about:blank` [Document](#) created when the [child browsing context](#) was created, then any [navigation](#) required of the user agent in that algorithm must be completed with [replacement enabled](#).

When a [Document](#) in a `<frame>` is marked as [completely loaded](#), the user agent must [queue a task](#) to [fire a simple event](#) named `load` at the `<frame>` element.

The [task source](#) for the [tasks](#) above is the [DOM manipulation task source](#).

When a `<frame>` element's [nested browsing context](#)'s [active document](#) is not [ready for post-load tasks](#), and when anything is [delaying the load event](#) of the `<frame>` element's [browsing context](#)'s [active document](#), and when the `<frame>` element's [browsing context](#) is in the [delaying load events mode](#), the `<frame>` must [delay the load event](#) of its document.

When the browsing context is created, if a `name` content attribute is present, the [browsing context name](#) must be set to the value of this attribute; otherwise, the [browsing context name](#) must be set to the empty string.

Whenever the `name` attribute is set, the nested [browsing context](#)'s [name](#) must be changed to the new value. If the attribute is removed, the [browsing context name](#) must be set to the empty string.

The `<frame>` element must implement the [HTMLFrameElement](#) interface.

```

interface HTMLFrameElement : HTMLElement {
    attribute DOMString name;
    attribute DOMString scrolling;
    attribute DOMString src;
    attribute DOMString frameBorder;
    attribute boolean noResize;
    readonly attribute Document? contentDocument;
    readonly attribute WindowProxy? contentWindow;

    [TreatNullAs=EmptyString] attribute DOMString marginHeight;
    [TreatNullAs=EmptyString] attribute DOMString marginWidth;
};

```

The **name**, **scrolling**, and **src** IDL attributes of the `<frame>` element must [reflect](#) the respective content attributes of the same name. For the purposes of reflection, the `<frame>` element's **src** content attribute is defined as containing a [URL](#).

The **frameBorder** IDL attribute of the `<frame>` element must [reflect](#) the element's **frameborder** content attribute.

The **noResize** IDL attribute of the `<frame>` element must [reflect](#) the element's **noresize** content attribute.

The **contentDocument** IDL attribute of the `<frame>` element must return the [Document](#) object of the [active document](#) of the `<frame>` element's [nested browsing context](#), if any and if its [origin](#) is the [same origin-domain](#) as the [origin](#) specified by the [incumbent settings object](#), or null otherwise.

The **contentWindow** IDL attribute must return the [WindowProxy](#) object of the `<frame>` element's [nested browsing context](#).

The **marginHeight** IDL attribute of the `<frame>` element must [reflect](#) the element's **marginheight** content attribute.

The **marginWidth** IDL attribute of the `<frame>` element must [reflect](#) the element's **marginwidth** content attribute.

§ 11.3.4. Application caches

An **application cache** is a set of cached resources consisting of:

- One or more resources (including their out-of-band metadata, such as HTTP headers, if any), identified by URLs, each falling into one (or more) of the following categories:

Master entries

NOTE:

These are documents that were added to the cache because a [browsing context](#) was [navigated](#) to that document and the document indicated that this was its cache, using the **manifest** attribute.

The manifest

NOTE:

This is the resource corresponding to the URL that was given in a master entry's `<html>` element's **manifest** attribute. The manifest is fetched and processed during the [application cache download process](#). All the [master entries](#) have the [same origin](#) as the manifest.

Explicit entries

NOTE:

These are the resources that were listed in the cache's [manifest](#) in an [explicit section](#).

Fallback entries

NOTE:

These are the resources that were listed in the cache's [manifest](#) in a [fallback section](#).

[Explicit entries](#) and [Fallback entries](#) can be marked as **foreign**, which means that they have a **manifest** attribute but that it doesn't point at this cache's [manifest](#).

NOTE:

A URL in the list can be flagged with multiple different types, and thus an entry can end up being categorized as multiple entries. For example, an entry can be a manifest entry and an explicit entry at the same time, if the manifest is listed within the manifest.

- Zero or more **fallback namespaces**, each of which is mapped to a [fallback entry](#).

NOTE:

These are URLs used as [prefix match patterns](#) for resources that are to be fetched from the network if possible, or to be replaced by the corresponding [fallback entry](#) if not. Each namespace URL has the [same origin](#) as [the manifest](#).

- Zero or more URLs that form the **online safelist namespaces**.

NOTE:

These are used as prefix match patterns, and declare URLs for which the user agent will ignore the application cache, instead fetching them normally (i.e., from the network or local HTTP cache as appropriate).

- An **online safelist wildcard flag**, which is either *open* or *blocking*.

NOTE:

The open state indicates that any URL not listed as cached is to be implicitly treated as being in the [online safelist namespaces](#); the blocking state indicates that URLs not listed explicitly in the manifest are to be treated as unavailable.

- A **cache mode flag**, which is either in the *fast* state or the *prefer-online* state.

Each [application cache](#) has a **completeness flag**, which is either *complete* or *incomplete*.



An **application cache group** is a group of [application caches](#), identified by the [absolute URL](#) of a resource [manifest](#) which is used to populate the caches in the group.

An [application cache](#) is **newer** than another if it was created after the other (in other words, [application caches](#) in an [application cache group](#) have a chronological order).

Only the newest [application cache](#) in an [application cache group](#) can have its [completeness flag](#) set to *incomplete*; the others are always all *complete*.

Each [application cache group](#) has an **update status**, which is one of the following: *idle*, *checking*, *downloading*.

A **relevant application cache** is an [application cache](#) that is the [newest](#) in its [group](#) to be *complete*.

Each [application cache group](#) has a **list of pending master entries**. Each entry in this list consists of a resource and a corresponding [Document](#) object. It is used during the [application cache download process](#) to ensure that new master entries are cached even if the [application cache download process](#) was already running for their [application cache group](#) when they were loaded.

An [application cache group](#) can be marked as **obsolete**, meaning that it must be ignored when looking at what [application cache groups](#) exist.



A **cache host** is a [Document](#) or a `SharedWorkerGlobalScope` object. A [cache host](#) can be associated with an [application cache](#).

[WEBWORKERS]

A [Document](#) initially is not associated with an [application cache](#), but can become associated with one early during the page

load process, when steps [in the parser](#) and in the [navigation](#) sections cause [cache selection](#) to occur.

A `SharedWorkerGlobalScope` can be associated with an [application cache](#) when it is created.

[WEBWORKERS]

Each [cache host](#) has an associated `ApplicationCache` object.



Multiple [application caches](#) in different [application cache groups](#) can contain the same resource, e.g., if the manifests all reference that resource. If the user agent is to **select an application cache** from a list of [relevant application caches](#) that contain a resource, the user agent must use the application cache that the user most likely wants to see the resource from, taking into account the following:

- which application cache was most recently updated,
- which application cache was being used to display the resource from which the user decided to look at the new resource, and
- which application cache the user prefers.



A URL **matches a fallback namespace** if there exists a [relevant application cache](#) whose [manifest](#)'s URL has the [same origin](#) as the URL in question, and that has a [fallback namespace](#) that is a [prefix match](#) for the URL being examined. If multiple fallback namespaces match the same URL, the longest one is the one that matches. A URL looking for a fallback namespace can match more than one application cache at a time, but only matches one namespace in each cache.

11.3.4.1. Parsing cache manifests

When a user agent is to **parse a manifest**, it means that the user agent must run the following steps:

1. [UTF-8 decode](#) the byte stream corresponding with the manifest to be parsed.

NOTE:

The [UTF-8 decode](#) algorithm strips a leading BOM, if any.

2. Let *base URL* be the [absolute URL](#) representing the manifest.
3. Apply the [URL parser](#) to *base URL*, and let *manifest path* be the [path](#) component thus obtained.
4. Remove all the characters in *manifest path* after the last U+002F SOLIDUS character (/), if any. (The first character and the last character in *manifest path* after this step will both be slashes, the URL path separator character.)
5. Apply the [URL parser](#) steps to the *base URL*, so that the components from its [URL record](#) can be used by the subsequent steps of this algorithm.
6. Let *explicit URLs* be an initially empty list of [absolute URLs](#) for [explicit entries](#).
7. Let *fallback URLs* be an initially empty mapping of [fallback namespaces](#) to [absolute URLs](#) for [fallback entries](#).
8. Let *online safelist namespaces* be an initially empty list of [absolute URLs](#) for an [online safelist](#).
9. Let *online safelist wildcard flag* be *blocking*.
10. Let *cache mode flag* be *fast*.
11. Let *input* be the decoded text of the manifest's byte stream.
12. Let *position* be a pointer into *input*, initially pointing at the first character.

13. If the characters starting from *position* are "CACHE", followed by a U+0020 SPACE character, followed by "MANIFEST", then advance *position* to the next character after those. Otherwise, this isn't a cache manifest; abort this algorithm with a failure while checking for the magic signature.
14. If the character at *position* is neither a U+0020 SPACE character, a U+0009 CHARACTER TABULATION (tab) character, U+000A LINE FEED (LF) character, nor a U+000D CARRIAGE RETURN (CR) character, then this isn't a cache manifest; abort this algorithm with a failure while checking for the magic signature.
15. This is a cache manifest. The algorithm cannot fail beyond this point (though bogus lines can get ignored).
16. [Collect a sequence of characters](#) that are *not* U+000A LINE FEED (LF) or U+000D CARRIAGE RETURN (CR) characters, and ignore those characters. (Extra text on the first line, after the signature, is ignored.)
17. Let *mode* be "explicit".
18. *Start of line*: If *position* is past the end of *input*, then jump to the last step. Otherwise, [collect a sequence of characters](#) that are U+000A LINE FEED (LF), U+000D CARRIAGE RETURN (CR), U+0020 SPACE, or U+0009 CHARACTER TABULATION (tab) characters.
19. Now, [collect a sequence of characters](#) that are *not* U+000A LINE FEED (LF) or U+000D CARRIAGE RETURN (CR) characters, and let the result be *line*.
20. Drop any trailing U+0020 SPACE and U+0009 CHARACTER TABULATION (tab) characters at the end of *line*.
21. If *line* is the empty string, then jump back to the step labeled *start of line*.
22. If the first character in *line* is a U+0023 NUMBER SIGN character (#), then jump back to the step labeled *Start of line*.
23. If *line* equals "CACHE:" (the word "CACHE" followed by a U+003A COLON character (:)), then set *mode* to "explicit" and jump back to the step labeled *Start of line*.
24. If *line* equals "FALLBACK:" (the word "FALLBACK" followed by a U+003A COLON character (:)), then set *mode* to "fallback" and jump back to the step labeled *Start of line*.
25. If *line* equals "NETWORK:" (the word "NETWORK" followed by a U+003A COLON character (:)), then set *mode* to "online safelist" and jump back to the step labeled *Start of line*.
26. If *line* equals "SETTINGS:" (the word "SETTINGS" followed by a U+003A COLON character (:)), then set *mode* to "settings" and jump back to the step labeled *Start of line*.
27. If *line* ends with a U+003A COLON character (:), then set *mode* to "unknown" and jump back to the step labeled *Start of line*.
28. This is either a data line or it is syntactically incorrect.
29. Let *position* be a pointer into *line*, initially pointing at the start of the string.
30. Let *tokens* be a list of strings, initially empty.
31. While *position* doesn't point past the end of *line*:
 1. Let *current token* be an empty string.
 2. While *position* doesn't point past the end of *line* and the character at *position* is neither a U+0020 SPACE nor a U+0009 CHARACTER TABULATION (tab) character, add the character at *position* to *current token* and advance *position* to the next character in *input*.
 3. Add *current token* to the *tokens* list.
 4. While *position* doesn't point past the end of *line* and the character at *position* is either a U+0020 SPACE or a U+0009 CHARACTER TABULATION (tab) character, advance *position* to the next character in *input*.
32. Process *tokens* as follows:

↪ If *mode* is "explicit"

Let *urlRecord* be the result of [parsing](#) the first item in *tokens*, with *base URL*; ignore the rest.

If *urlRecord* is failure, then jump back to the step labeled *Start of line*.

If *urlRecord* has a different [scheme](#) component than *base URL* (the manifest's URL), then jump back to the step labeled *Start of line*.

Let *new URL* be the result of applying the [URL serializer](#) algorithm to *urlRecord*, with the *exclude fragment flag* set.

Add *new URL* to the *explicit URLs*.

↪ If *mode* is "fallback"

Let *part one* be the first token in *tokens*, and let *part two* be the second token in *tokens*.

Let *urlRecordOne* be the result of [parsing](#) *part one* with *base URL*.

Let *urlRecordTwo* be the result of [parsing](#) *part two* with *base URL*.

If either *urlRecordOne* or *urlRecordTwo* is failure, then jump back to the step labeled *Start of line*.

If the [origin](#) of either *urlRecordOne* or *urlRecordTwo* is not [same origin](#) with the manifest's URL [origin](#), then jump back to the step labeled *Start of line*.

Let *part one path* be the [path](#) component of *urlRecordOne*.

If *manifest path* is not a [prefix match](#) for *part one path*, then jump back to the step labeled *Start of line*.

Let *part one* be the result of applying the [URL serializer](#) algorithm to *urlRecordOne*, with the *exclude fragment flag* set.

Let *part two* be the result of applying the [URL serializer](#) algorithm to *urlRecordTwo*, with the *exclude fragment flag* set.

If *part one* is already in the *fallback URLs* mapping as a [fallback namespace](#), then jump back to the step labeled *Start of line*.

Otherwise, add *part one* to the *fallback URLs* mapping as a [fallback namespace](#), mapped to *part two* as the [fallback entry](#).

↪ If *mode* is "online safelist"

If the first item in *tokens* is a U+002A ASTERISK character (*), then set *online safelist wildcard flag* to *open* and jump back to the step labeled *Start of line*.

Otherwise, let *urlRecord* be the result of [parsing](#) the first item in *tokens* with *base URL*.

If *urlRecord* is failure, then jump back to the step labeled *Start of line*.

If *urlRecord* has a different [scheme](#) component than *base URL* (the manifest's URL), then jump back to the step labeled *Start of line*.

Let *new URL* be the result of applying the [URL serializer](#) algorithm to *urlRecord*, with the *exclude fragment flag* set.

Add *new URL* to the *online safelist namespaces*.

↪ If *mode* is "settings"

If *tokens* contains a single token, and that token is a [case-sensitive](#) match for the string "prefer-online", then set *cache mode flag* to *prefer-online* and jump back to the step labeled *Start of line*.

Otherwise, the line is an unsupported setting: do nothing; the line is ignored.

↪ If *mode* is "unknown"

Do nothing. The line is ignored.

33. Jump back to the step labeled *Start of line*. (That step jumps to the next, and last, step when the end of the file is

reached.)

34. Return the *explicit URLs* list, the *fallback URLs* mapping, the *online safelist namespaces*, the *online safelist wild-card flag*, and the *cache mode flag*.

§ 11.3.4.2. Downloading or updating an application cache

When the user agent is required (by other parts of this specification) to start the **application cache download process** for an *absolute URL* purported to identify a *manifest*, or for an *application cache group*, potentially given a particular *cache host*, and potentially given a *master* resource, the user agent must run the steps below. These steps are always run *in parallel* with the *event loop tasks*.

Some of these steps have requirements that only apply if the user agent **shows caching progress**. Support for this is optional. Certain events fired during the *application cache download process* allow the script to override the display of such an interface. (Such events are delayed until after the *load* event has fired.)

User agents are encouraged not to show prominent update progress notifications for applications that cancel the relevant events.

The *application cache download process* steps are as follows:

1. Optionally, wait until the permission to start the *application cache download process* has been obtained from the user and until the user agent is confident that the network is available. This could include doing nothing until the user explicitly opts-in to caching the site, or could involve prompting the user for permission. The algorithm might never get past this point. (This step is particularly intended to be used by user agents running on severely space-constrained devices or in highly privacy-sensitive environments).
2. Atomically, so as to avoid race conditions, perform the following substeps:
 1. Pick the appropriate substeps:
 - ↪ If these steps were invoked with an *absolute URL* purported to identify a *manifest*

Let *manifest URL* be that *absolute URL*.

If there is no *application cache group* identified by *manifest URL*, then create a new *application cache group* identified by *manifest URL*. Initially, it has no *application caches*. One will be created later in this algorithm.
 - ↪ If these steps were invoked with an *application cache group*

Let *manifest URL* be the *absolute URL* of the *manifest* used to identify the *application cache group* to be updated.

If that *application cache group* is *obsolete*, then abort this instance of the *application cache download process*. This can happen if another instance of this algorithm found the manifest to be 404 or 410 while this algorithm was waiting in the first step above.
 2. Let *cache group* be the *application cache group* identified by *manifest URL*.
 3. If these steps were invoked with a *master* resource, then add the resource, along with the resource's *Document*, to *cache group*'s *list of pending master entries*.
 4. If these steps were invoked with a *cache host*, and the *status* of *cache group* is *checking* or *downloading*, then *queue a post-load task to fire a simple event* named *checking* that is cancelable at the *ApplicationCache* singleton of that *cache host*. The default action of this event must be, if the user agent **shows caching progress**, the display of some sort of user interface indicating to the user that the user agent is checking to see if it can download the application.
 5. If these steps were invoked with a *cache host*, and the *status* of *cache group* is *downloading*, then also *queue a post-load task to fire a simple event* named *downloading* that is cancelable at the *ApplicationCache* singleton of that *cache host*. The default action of this event must be, if the user agent **shows caching progress**, the display of some sort of user interface indicating to the user the application is being downloaded.

6. If the [status](#) of the `cache group` is either *checking* or *downloading*, then abort this instance of the [application cache download process](#), as an update is already in progress.
7. Set the [status](#) of `cache group` to *checking*.
8. For each [cache host](#) associated with an [application cache](#) in `cache group`, [queue a post-load task to fire a simple event](#) that is cancelable named *checking* at the `ApplicationCache` singleton of the [cache host](#). The default action of these events must be, if the user agent [shows caching progress](#), the display of some sort of user interface indicating to the user that the user agent is checking for the availability of updates.

NOTE:

The remainder of the steps run [in parallel](#).

If `cache group` already has an [application cache](#) in it, then this is an **upgrade attempt**. Otherwise, this is a **cache attempt**.

3. If this is a [cache attempt](#), then this algorithm was invoked with a [cache host](#); [queue a post-load task to fire a simple event](#) named *checking* that is cancelable at the `ApplicationCache` singleton of that [cache host](#). The default action of this event must be, if the user agent [shows caching progress](#), the display of some sort of user interface indicating to the user that the user agent is checking for the availability of updates.
4. Let *request* be a new [request](#) whose [URL](#) is *manifest URL*, [client](#) is null, [destination](#) is "subresource", [omit-Origin-header flag](#) is set, [referrer](#) is "no-referrer", [synchronous flag](#) is set, [credentials mode](#) is "include", and whose [use-URL-credentials flag](#) is set.
5. *Fetching the manifest*: Let *manifest* be the result of [fetching request](#). HTTP caching semantics should be honored for this request.

Parse *manifest*'s [body](#) according to the [rules for parsing manifests](#), obtaining a list of [explicit entries](#), [fallback entries](#) and the [fallback namespaces](#) that map to them, entries for the [online safelist](#), and values for the [online safelist wildcard flag](#) and the [cache mode flag](#).

NOTE:

The [MIME type](#) of the resource is ignored — it is assumed to be `text/cache-manifest`. In the future, if new manifest formats are supported, the different types will probably be distinguished on the basis of the file signatures (for the current format, that is the "CACHE MANIFEST" string at the top of the file).

6. If *fetching the manifest* fails due to a 404 or 410 response status, then run these substeps:
 1. Mark `cache group` as [obsolete](#). This `cache group` no longer exists for any purpose other than the processing of [Document](#) objects already associated with an [application cache](#) in the `cache group`.
 2. Let *task list* be an empty list of [tasks](#).
 3. For each [cache host](#) associated with an [application cache](#) in `cache group`, create a [task](#) to [fire a simple event](#) named *obsolete* that is cancelable at the `ApplicationCache` singleton of the [cache host](#), and append it to *task list*. The default action of these events must be, if the user agent [shows caching progress](#), the display of some sort of user interface indicating to the user that the application is no longer available for offline use.
 4. For each entry in `cache group`'s [list of pending master entries](#), create a [task](#) to [fire a simple event](#) that is cancelable named *error* (not *obsolete*!) at the `ApplicationCache` singleton of the [Document](#) for this entry, if there still is one, and append it to *task list*. The default action of this event must be, if the user agent [shows caching progress](#), the display of some sort of user interface indicating to the user that the user agent failed to save the application for offline use.
 5. If `cache group` has an [application cache](#) whose [completeness flag](#) is *incomplete*, then discard that [application cache](#).
 6. If appropriate, remove any user interface indicating that an update for this cache is in progress.
 7. Let the [status](#) of `cache group` be *idle*.

8. For each [task](#) in [task list](#), [queue that task as a post-load task](#).
9. Abort the [application cache download process](#).
7. Otherwise, if *fetching the manifest* fails in some other way (e.g., the server returns another 4xx or 5xx response, or there is a DNS error, or the connection times out, or the user cancels the download, or the parser for manifests fails when checking the magic signature), or if the server returned a redirect, then run the [cache failure steps](#). [\[HTTP\]](#)
8. If this is an [upgrade attempt](#) and the newly downloaded [manifest](#) is byte-for-byte identical to the manifest found in the [newest application cache](#) in [cache group](#), or the response status is 304, then run these substeps:

1. Let [cache](#) be the [newest application cache](#) in [cache group](#).
2. Let [task list](#) be an empty list of [tasks](#).
3. For each entry in [cache group](#)'s [list of pending master entries](#), wait for the resource for this entry to have either completely downloaded or failed.

If the download failed (e.g., the server returns a 4xx or 5xx response, or there is a DNS error, the connection times out, or the user cancels the download), or if the resource is labeled with the "no-store" cache directive, then create a [task to fire a simple event](#) that is cancelable named `error` at the `ApplicationCache` singleton of the [Document](#) for this entry, if there still is one, and append it to [task list](#). The default action of this event must be, if the user agent [shows caching progress](#), the display of some sort of user interface indicating to the user that the user agent failed to save the application for offline use.

Otherwise, associate the [Document](#) for this entry with [cache](#); store the resource for this entry in [cache](#), if it isn't already there, and categorize its entry as a [master entry](#). If applying the [URL parser](#) algorithm to the resource's [URL](#) results in a [resulting URL record](#) that has a non-null [fragment](#) component, the [URL](#) used for the entry in [cache](#) must instead be the [absolute URL](#) obtained from applying the [URL serializer](#) algorithm to the [resulting URL record](#) with the [exclude fragment flag](#) set (application caches never include fragment identifiers).

4. For each [cache host](#) associated with an [application cache](#) in [cache group](#), create a [task to fire a simple event](#) that is cancelable named `noupdate` at the `ApplicationCache` singleton of the [cache host](#), and append it to [task list](#). The default action of these events must be, if the user agent [shows caching progress](#), the display of some sort of user interface indicating to the user that the application is up to date.
5. Empty [cache group](#)'s [list of pending master entries](#).
6. If appropriate, remove any user interface indicating that an update for this cache is in progress.
7. Let the [status](#) of [cache group](#) be *idle*.
8. For each [task](#) in [task list](#), [queue that task as a post-load task](#).
9. Abort the [application cache download process](#).
9. Let [new cache](#) be a newly created [application cache](#) in [cache group](#). Set its [completeness flag](#) to *incomplete*.
10. For each entry in [cache group](#)'s [list of pending master entries](#), associate the [Document](#) for this entry with [new cache](#).
11. Set the [status](#) of [cache group](#) to *downloading*.
12. For each [cache host](#) associated with an [application cache](#) in [cache group](#), [queue a post-load task to fire a simple event](#) that is cancelable named `downloading` at the `ApplicationCache` singleton of the [cache host](#). The default action of these events must be, if the user agent [shows caching progress](#), the display of some sort of user interface indicating to the user that a new version is being downloaded.
13. Let [file list](#) be an empty list of URLs with flags.
14. Add all the URLs in the list of [explicit entries](#) obtained by parsing [manifest](#) to [file list](#), each flagged with "explicit entry".
15. Add all the URLs in the list of [fallback entries](#) obtained by parsing [manifest](#) to [file list](#), each flagged with "fallback entry".

16. If this is an [upgrade attempt](#), then add all the URLs of [master entries](#) in the [newest application cache](#) in *cache group* whose [completeness flag](#) is *complete* to *file list*, each flagged with "master entry".
17. If any URL is in *file list* more than once, then merge the entries into one entry for that URL, that entry having all the flags that the original entries had.
18. For each URL in *file list*, run the following steps. These steps may be run in parallel for two or more of the URLs at a time. If, while running these steps, the `ApplicationCache` object's `abort()` method [sends a signal](#) to this instance of the [application cache download process](#) algorithm, then run the [cache failure steps](#) instead.
 1. If the resource URL being processed was flagged as neither an "explicit entry" nor or a "fallback entry", then the user agent may skip this URL.

NOTE:

This is intended to allow user agents to expire resources not listed in the manifest from the cache. Generally, implementors are urged to use an approach that expires lesser-used resources first.

2. For each [cache host](#) associated with an [application cache](#) in *cache group*, [queue a progress post-load task](#) to [fire](#) a [trusted](#) event with the name `progress`, which does not bubble, which is cancelable, and which uses the `ProgressEvent` interface, at the `ApplicationCache` singleton of the [cache host](#). The `lengthComputable` attribute must be set to `true`, the `total` attribute must be set to the number of files in *file list*, and the `loaded` attribute must be set to the number of files in *file list* that have been either downloaded or skipped so far. The default action of these events must be, if the user agent [shows caching progress](#), the display of some sort of user interface indicating to the user that a file is being downloaded in preparation for updating the application. [\[XHR\]](#)
3. Let *request* be a new [request](#) whose [URL](#) is URL, [client](#) is null, [destination](#) is "subresource", [origin](#) is *manifest URL*'s [origin](#), [referrer](#) is "no-referrer", [synchronous flag](#) is set, [credentials mode](#) is "include", [use-URL-credentials flag](#) is set, and [redirect mode](#) is "manual".
4. [Fetch](#) *request*. If this is an [upgrade attempt](#), then use the [newest application cache](#) in *cache group* as an HTTP cache, and honor HTTP caching semantics (such as expiration, ETags, and so forth) with respect to that cache. User agents may also have other caches in place that are also honored.
5. If the previous step fails (e.g., the server returns a 4xx or 5xx response, or there is a DNS error, or the connection times out, or the user cancels the download), or if the server returned a redirect, or if the resource is labeled with the "no-store" cache directive, then run the first appropriate step from the following list: [\[HTTP\]](#)

↪ **If the URL being processed was flagged as an "explicit entry" or a "fallback entry"**

If these steps are being run in parallel for any other URLs in *file list*, then abort these steps for those other URLs. Run the [cache failure steps](#).

NOTE:

Redirects are fatal because they are either indicative of a network problem (e.g., a captive portal); or would allow resources to be added to the cache under URLs that differ from any URL that the networking model will allow access to, leaving orphan entries; or would allow resources to be stored under URLs different than their true URLs. All of these situations are bad.

↪ **If the error was a 404 or 410 HTTP response**

↪ **If the resource was labeled with the "no-store" cache directive**

Skip this resource. It is dropped from the cache.

↪ **Otherwise**

Copy the resource and its metadata from the [newest application cache](#) in *cache group* whose [completeness flag](#) is *complete*, and act as if that was the fetched resource, ignoring the resource obtained from the network.

NOTE:

These rules make errors for resources listed in the manifest fatal, while making it possible for other resources to be removed from caches when they are removed from the server, without errors, and making non-manifest resources survive server-side errors.

NOTE:

Except for the "no-store" directive, HTTP caching rules that would cause a file to be expired or otherwise not cached are ignored for the purposes of the [application cache download process](#).

6. Otherwise, the fetching succeeded. Store the resource in the *new cache*.

If the user agent is not able to store the resource (e.g., because of quota restrictions), the user agent may prompt the user or try to resolve the problem in some other manner (e.g., automatically pruning content in other caches).

If the problem cannot be resolved, the user agent must run the [cache failure steps](#).

7. If the URL being processed was flagged as an "explicit entry" in *file list*, then categorize the entry as an [explicit entry](#).
8. If the URL being processed was flagged as a "fallback entry" in *file list*, then categorize the entry as a [fallback entry](#).
9. If the URL being processed was flagged as an "master entry" in *file list*, then categorize the entry as a [master entry](#).
10. As an optimization, if the resource is an HTML or XML file whose root element is an `<html>` element with a `manifest` attribute whose value doesn't match the manifest URL of the application cache being processed, then the user agent should mark the entry as being [foreign](#).
19. For each [cache host](#) associated with an [application cache](#) in *cache group*, [queue a progress post-load task](#) to [fire](#) a [trusted](#) event with the name `progress`, which does not bubble, which is cancelable, and which uses the `ProgressEvent` interface, at the `ApplicationCache` singleton of the [cache host](#). The `lengthComputable` attribute must be set to `true`, the `total` and the `loaded` attributes must be set to the number of files in *file list*. The default action of these events must be, if the user agent [shows caching progress](#), the display of some sort of user interface indicating to the user that all the files have been downloaded. [\[XHR\]](#)
20. Store the list of [fallback namespaces](#), and the URLs of the [fallback entries](#) that they map to, in *new cache*.
21. Store the URLs that form the new [online safelist](#) in *new cache*.
22. Store the value of the new [online safelist wildcard flag](#) in *new cache*.
23. Store the value of the new [cache mode flag](#) in *new cache*.
24. For each entry in *cache group*'s [list of pending master entries](#), wait for the resource for this entry to have either completely downloaded or failed.

If the download failed (e.g., the server returns a 4xx or 5xx response, or there is a DNS error, the connection times out, or the user cancels the download), or if the resource is labeled with the "no-store" cache directive, then run these sub-steps:

1. Unassociate the [Document](#) for this entry from *new cache*.
2. [Queue a post-load task](#) to [fire a simple event](#) that is cancelable named `error` at the `ApplicationCache` singleton of the [Document](#) for this entry, if there still is one. The default action of this event must be, if the user agent [shows caching progress](#), the display of some sort of user interface indicating to the user that the user agent failed to save the application for offline use.
3. If this is a [cache attempt](#) and this entry is the last entry in *cache group*'s [list of pending master entries](#), then run these further sub-steps:
 1. Discard *cache group* and its only [application cache](#), *new cache*.
 2. If appropriate, remove any user interface indicating that an update for this cache is in progress.
 3. Abort the [application cache download process](#).
4. Otherwise, remove this entry from *cache group*'s [list of pending master entries](#).

Otherwise, store the resource for this entry in *new cache*, if it isn't already there, and categorize its entry as a [master](#)

[entry](#).

25. Let *request* be a new [request](#) whose [URL](#) is *manifest URL*, [client](#) is null, [destination](#) is "subresource", [referrer](#) is "no-referrer", [synchronous flag](#) is set, [credentials mode](#) is "include", and whose [use-URL-credentials flag](#) is set.
26. Let *second manifest* be the result of [fetching request](#). HTTP caching semantics should again be honored for this request.
27. If the previous step failed for any reason, or if the fetching attempt involved a redirect, or if *second manifest* and *manifest* are not byte-for-byte identical, then schedule a rerun of the entire algorithm with the same parameters after a short delay, and run the [cache failure steps](#).
28. Otherwise, store *manifest* in *new cache*, if it's not there already, and categorize its entry as [the manifest](#).
29. Set the [completeness flag](#) of *new cache* to *complete*.
30. Let *task list* be an empty list of [tasks](#).
31. If this is a [cache attempt](#), then for each [cache host](#) associated with an [application cache](#) in *cache group*, create a [task](#) to [fire a simple event](#) that is cancelable named *cached* at the *ApplicationCache* singleton of the [cache host](#), and append it to *task list*. The default action of these events must be, if the user agent [shows caching progress](#), the display of some sort of user interface indicating to the user that the application has been cached and that they can now use it offline.

Otherwise, it is an [upgrade attempt](#). For each [cache host](#) associated with an [application cache](#) in *cache group*, create a [task](#) to [fire a simple event](#) that is cancelable named *updateready* at the *ApplicationCache* singleton of the [cache host](#), and append it to *task list*. The default action of these events must be, if the user agent [shows caching progress](#), the display of some sort of user interface indicating to the user that a new version is available and that they can activate it by reloading the page.
32. If appropriate, remove any user interface indicating that an update for this cache is in progress.
33. Set the [update status](#) of *cache group* to *idle*.
34. For each [task](#) in *task list*, [queue that task as a post-load task](#).

The **cache failure steps** are as follows:

1. Let *task list* be an empty list of [tasks](#).
2. For each entry in *cache group*'s [list of pending master entries](#), run the following further substeps. These steps may be run in parallel for two or more entries at a time.
 1. Wait for the resource for this entry to have either completely downloaded or failed.
 2. Unassociate the [Document](#) for this entry from its [application cache](#), if it has one.
 3. Create a [task](#) to [fire a simple event](#) that is cancelable named *error* at the *ApplicationCache* singleton of the [Document](#) for this entry, if there still is one, and append it to *task list*. The default action of these events must be, if the user agent [shows caching progress](#), the display of some sort of user interface indicating to the user that the user agent failed to save the application for offline use.
3. For each [cache host](#) still associated with an [application cache](#) in *cache group*, create a [task](#) to [fire a simple event](#) that is cancelable named *error* at the *ApplicationCache* singleton of the [cache host](#), and append it to *task list*. The default action of these events must be, if the user agent [shows caching progress](#), the display of some sort of user interface indicating to the user that the user agent failed to save the application for offline use.
4. Empty *cache group*'s [list of pending master entries](#).
5. If *cache group* has an [application cache](#) whose [completeness flag](#) is *incomplete*, then discard that [application cache](#).
6. If appropriate, remove any user interface indicating that an update for this cache is in progress.
7. Let the [status](#) of *cache group* be *idle*.
8. If this was a [cache attempt](#), discard *cache group* altogether.

9. For each [task](#) in *task list*, [queue that task as a post-load task](#).

10. Abort the [application cache download process](#).

Attempts to fetch resources as part of the [application cache download process](#) may be done with cache-defeating semantics, to avoid problems with stale or inconsistent intermediary caches.



User agents may invoke the [application cache download process](#), in the background, for any [application cache group](#), at any time (with no [cache host](#)). This allows user agents to keep caches primed and to update caches even before the user visits a site.



Each [Document](#) has a list of **pending application cache download process tasks** that is used to delay events fired by the algorithm above until the document's load event has fired. When the [Document](#) is created, the list must be empty.

When the steps above say to **queue a post-load task** *task*, where *task* is a [task](#) that dispatches an event on a target [ApplicationCache](#) object *target*, the user agent must run the appropriate steps from the following list:

If *target*'s [node document](#) is ready for post-load tasks

[Queue](#) the task *task*.

Otherwise

Add *task* to *target*'s [node document](#)'s list of [pending application cache download process tasks](#).

When the steps above say to **queue a progress post-load task** *task*, where *task* is a [task](#) that dispatches an event on a target [ApplicationCache](#) object *target*, the user agent must run the following steps:

1. If there is a *task* in *target*'s [node document](#)'s list of [pending application cache download process tasks](#) that is labeled as a *progress task*, then remove that task from the list.
2. Label *task* as a *progress task*.
3. [Queue a post-load task](#) *task*.

The [task source](#) for these [tasks](#) is the [networking task source](#).

§ 11.3.4.3. The application cache selection algorithm

When the **application cache selection algorithm** is invoked with a [Document](#) *document* and optionally a manifest [URL](#) *manifest URL*, the user agent must run the first applicable set of steps from the following list:

↪ **If there is a *manifest URL*, and *document* was loaded from an [application cache](#), and the URL of the [manifest](#) of that cache's [application cache group](#) is not the same as *manifest URL***

Mark the entry for the resource from which *document* was taken in the [application cache](#) from which it was loaded as [foreign](#).

Restart the current navigation from the top of the [navigation algorithm](#), undoing any changes that were made as part of the initial load (changes can be avoided by ensuring that the step to [update the session history with the new page](#) is only ever completed *after* this [application cache selection algorithm](#) is run, though this is not required).

↪ **If *document* was loaded from an [application cache](#), and that [application cache](#) still exists (it is not now [obsolete](#))**

Associate *document* with the [application cache](#) from which it was loaded. Invoke, in the background, the [application cache download process](#) for that [application cache](#)'s [application cache group](#), with *document* as the [cache host](#).

↪ **If *document* was loaded using GET, and, there is a *manifest URL*, and *manifest URL* has the [same origin](#) as *document***

Invoke, in the background, the [application cache download process](#) for *manifest URL*, with *document* as the

[cache host](#) and with the resource from which *document* was parsed as the [master](#) resource.

If there are [relevant application caches](#) that are identified by a URL with the [same origin](#) as the URL of *document*, and that have this URL as one of their entries, excluding entries marked as [foreign](#), then the user agent should use the [most appropriate application cache](#) of those that match as an HTTP cache for any subresource loads. User agents may also have other caches in place that are also honored.

↪ Otherwise

The [Document](#) is not associated with any [application cache](#).

If there was a [manifest URL](#), the user agent may report to the user that it was ignored, to aid in application development.

§ 11.3.4.4. *Changes to the networking model*

When a [cache host](#) is associated with an [application cache](#) whose [completeness flag](#) is *complete*, any and all loads for resources related to that [cache host](#) other than those for [child browsing contexts](#) must go through the following steps instead of immediately invoking the mechanisms appropriate to that resource's scheme:

1. If the resource is not to be fetched using the GET method, or if applying the [URL parser](#) algorithm to both its [URL](#) and the [application cache's manifest's](#) URL results in two [URL records](#) with different [scheme](#) components, then fetch the resource normally and abort these steps.
2. If the resource's URL is a [master entry](#), the [manifest](#), an [explicit entry](#), or a [fallback entry](#) in the [application cache](#), then get the resource from the cache (instead of fetching it), and abort these steps.
3. If there is an entry in the [application cache's online safelist](#) that has the [same origin](#) as the resource's URL and that is a [prefix match](#) for the resource's URL, then fetch the resource normally and abort these steps.
4. If the resource's URL has the [same origin](#) as the manifest's URL, and there is a [fallback namespace](#) *f* in the [application cache](#) that is a [prefix match](#) for the resource's URL, then:

Fetch the resource normally. If this results in a redirect to a resource with another [origin](#) (indicative of a captive portal), or a 4xx or 5xx status code, or if there were network errors (but not if the user canceled the download), then instead get, from the cache, the resource of the [fallback entry](#) corresponding to the [fallback namespace](#) *f*. Abort these steps.

5. If the [application cache's online safelist wildcard flag](#) is *open*, then fetch the resource normally and abort these steps.
6. Fail the resource load as if there had been a generic network error.

NOTE:

The above algorithm ensures that so long as the [online safelist wildcard flag](#) is blocking, resources that are not present in the [manifest](#) will always fail to load (at least, after the [application cache](#) has been primed the first time), making the testing of offline applications simpler.

§ 11.3.4.5. *Expiring application caches*

As a general rule, user agents should not expire application caches, except on request from the user, or after having been left unused for an extended period of time.

Application caches and cookies have similar implications with respect to privacy (e.g., if the site can identify the user when providing the cache, it can store data in the cache that can be used for cookie resurrection). Implementors are therefore encouraged to expose application caches in a manner related to HTTP cookies, allowing caches to be expunged together with cookies and other origin-specific data.

§ 11.3.4.6. *Disk space*

User agents should consider applying constraints on disk usage of [application caches](#), and care should be taken to ensure that the restrictions cannot be easily worked around using subdomains.

User agents should allow users to see how much space each domain is using, and may offer the user the ability to delete specific [application caches](#).

§ 11.3.4.7. Security concerns with offline applications caches

This section is non-normative.

The main risk introduced by offline application caches is that an injection attack can be elevated into persistent site-wide page replacement. This attack involves using an injection vulnerability to upload two files to the victim site. The first file is an application cache manifest consisting of just a fallback entry pointing to the second file, which is an HTML page whose manifest is declared as that first file. Once the user has been directed to that second file, all subsequent accesses to any file covered by the given fallback namespace while either the user or the site is offline will instead show that second file. Targeted denial-of-service attacks or cookie bombing attacks (where the client is made to send so many cookies that the server refuses to process the request) can be used to ensure that the site appears offline.

To mitigate this, manifests can only specify fallbacks that are in the same path as the manifest itself. This means that a content injection upload vulnerability in a particular directory on a server can only be escalated to a take-over of that directory and its subdirectories. If there is no way to inject a file into the root directory, the entire site cannot be taken over.

If a site has been attacked in this way, simply removing the offending manifest might eventually clear the problem, since the next time the manifest is updated, a 404 error will be seen, and the user agent will clear the cache. "Eventually" is the key word here, however; while the attack on the user or server is ongoing, such that connections from an affected user to the affected site are blocked, the user agent will simply assume that the user is offline and will continue to use the hostile manifest. Unfortunately, if a cookie bombing attack has also been used, merely removing the manifest is insufficient; in addition, the server has to be configured to return a 404 or 410 response instead of the 413 "Request Entity Too Large" response.

TLS does not inherently protect a site from this attack, since the attack relies on content being served from the server itself. Not using application caches also does not prevent this attack, since the attack relies on an attacker-provided manifest.

§ 11.3.4.8. Application cache API

```
[Exposed=(Window, SharedWorker)]
interface ApplicationCache : EventTarget {
  // update status
  const unsigned short UNCACHED = 0;
  const unsigned short IDLE = 1;
  const unsigned short CHECKING = 2;
  const unsigned short DOWNLOADING = 3;
  const unsigned short UPDATEREADY = 4;
  const unsigned short OBSOLETE = 5;
  readonly attribute unsigned short status;

  // updates
  void update();
  void abort();
  void swapCache();

  // events
  attribute EventHandler onchecking;
  attribute EventHandler onerror;
  attribute EventHandler onnoupdate;
  attribute EventHandler ondownloading;
  attribute EventHandler onprogress;
  attribute EventHandler onupdateready;
  attribute EventHandler oncached;
  attribute EventHandler onobsolete;
};
```

This definition is non-normative. Implementation requirements are given below this definition.

```
cache = window.applicationCache
```

(In a window.) Returns the `ApplicationCache` object that applies to the [active document](#) of that [Window](#).

`cache = self.applicationCache`

(In a shared worker.) Returns the `ApplicationCache` object that applies to the current shared worker.

`cache . status`

Returns the current status of the application cache, as given by the constants defined below.

`cache . update()`

Invokes the [application cache download process](#).

Throws an `InvalidStateError` exception if there is no application cache to update.

`cache . abort()`

Cancels the [application cache download process](#).

`cache . swapCache()`

Switches to the most recent application cache, if there is a newer one. If there isn't, throws an `InvalidStateError` exception.

There is a one-to-one mapping from [cache hosts](#) to `ApplicationCache` objects. The `applicationCache` attribute on `Window` objects must return the `ApplicationCache` object associated with the `Window` object's [active document](#). The `applicationCache` attribute on `SharedWorkerGlobalScope` objects must return the `ApplicationCache` object associated with the worker.

NOTE:

A `Window` or `SharedWorkerGlobalScope` object has an associated `ApplicationCache` object even if that [cache host](#) has no actual [application cache](#).



The `status` attribute, on getting, must return the current state of the [application cache](#) that the `ApplicationCache` object's [cache host](#) is associated with, if any. This must be the appropriate value from the following list:

UNCACHED (numeric value 0)

The `ApplicationCache` object's [cache host](#) is not associated with an [application cache](#) at this time.

IDLE (numeric value 1)

The `ApplicationCache` object's [cache host](#) is associated with an [application cache](#) whose [application cache group](#)'s [update status](#) is *idle*, and that [application cache](#) is the *newest* cache in its [application cache group](#), and the [application cache group](#) is not marked as *obsolete*.

CHECKING (numeric value 2)

The `ApplicationCache` object's [cache host](#) is associated with an [application cache](#) whose [application cache group](#)'s [update status](#) is *checking*.

DOWNLOADING (numeric value 3)

The `ApplicationCache` object's [cache host](#) is associated with an [application cache](#) whose [application cache group](#)'s [update status](#) is *downloading*.

UPDATEREADY (numeric value 4)

The `ApplicationCache` object's [cache host](#) is associated with an [application cache](#) whose [application cache group](#)'s [update status](#) is *idle*, and whose [application cache group](#) is not marked as *obsolete*, but that [application cache](#) is *not* the *newest* cache in its group.

OBSOLETE (numeric value 5)

The `ApplicationCache` object's [cache host](#) is associated with an [application cache](#) whose [application cache group](#) is marked as *obsolete*.



If the **update()** method is invoked, the user agent must invoke the [application cache download process](#), in the background, for the [application cache group](#) of the [application cache](#) with which the `ApplicationCache` object's [cache host](#) is associated, but without giving that [cache host](#) to the algorithm. If there is no such [application cache](#), or if its [application cache group](#) is marked as [obsolete](#), then the method must throw an `InvalidStateError` exception instead.

If the **abort()** method is invoked, the user agent must **send a signal** to the current [application cache download process](#) for the [application cache group](#) of the [application cache](#) with which the `ApplicationCache` object's [cache host](#) is associated, if any. If there is no such [application cache](#), or it does not have a current [application cache download process](#), then do nothing.

If the **swapCache()** method is invoked, the user agent must run the following steps:

1. Check that `ApplicationCache` object's [cache host](#) is associated with an [application cache](#). If it is not, then throw an `InvalidStateError` exception and abort these steps.
2. Let *cache* be the [application cache](#) with which the `ApplicationCache` object's [cache host](#) is associated. (By definition, this is the same as the one that was found in the previous step.)
3. If *cache*'s [application cache group](#) is marked as [obsolete](#), then unassociate the `ApplicationCache` object's [cache host](#) from *cache* and abort these steps. (Resources will now load from the network instead of the cache.)
4. Check that there is an application cache in the same [application cache group](#) as *cache* whose [completeness flag](#) is *complete* and that is [newer](#) than *cache*. If there is not, then throw an `InvalidStateError` exception and abort these steps.
5. Let *new cache* be the [newest application cache](#) in the same [application cache group](#) as *cache* whose [completeness flag](#) is *complete*.
6. Unassociate the `ApplicationCache` object's [cache host](#) from *cache* and instead associate it with *new cache*.

The following are the [event handlers](#) (and their corresponding [event handler event types](#)) that must be supported, as [event handler IDL attributes](#), by all objects implementing the `ApplicationCache` interface:

Event handler	Event handler event type
onchecking	checking
onerror	error
onnoupdate	noupdate
ondownloading	downloading
onprogress	progress
onupdateready	updateready
oncached	cached
onobsolete	obsolete

§ 11.3.5. Other elements, attributes and APIs

User agents must treat [<acronym>](#) elements in a manner equivalent to [<abbr>](#) elements in terms of semantics and for purposes of rendering.



```
partial interface HTMLAnchorElement {
  attribute DOMString coords;
  attribute DOMString charset;
  attribute DOMString name;
  attribute DOMString shape;
};
```

The **coords**, **charset**, **name**, and **shape** IDL attributes of the `<th>` element must [reflect](#) the respective content attributes of the same name.



```
partial interface HTMLAreaElement {
  attribute boolean noHref;
};
```

The **noHref** IDL attribute of the `<area>` element must [reflect](#) the element's **nohref** content attribute.



```
partial interface HTMLBodyElement {
  [TreatNullAs=EmptyString] attribute DOMString text;
  [TreatNullAs=EmptyString] attribute DOMString link;
  [TreatNullAs=EmptyString] attribute DOMString vLink;
  [TreatNullAs=EmptyString] attribute DOMString aLink;
  [TreatNullAs=EmptyString] attribute DOMString bgColor;
  attribute DOMString background;
};
```

The **text** IDL attribute of the `<body>` element must [reflect](#) the element's **text** content attribute.

The **link** IDL attribute of the `<body>` element must [reflect](#) the element's **link** content attribute.

The **aLink** IDL attribute of the `<body>` element must [reflect](#) the element's **aLink** content attribute.

The **vLink** IDL attribute of the `<body>` element must [reflect](#) the element's **vLink** content attribute.

The **bgColor** IDL attribute of the `<body>` element must [reflect](#) the element's **bgcolor** content attribute.

The **background** IDL attribute of the `<body>` element must [reflect](#) the element's **background** content attribute. (The **background** content is *not* defined to contain a [URL](#), despite rules regarding its handling in [§10 Rendering](#) above.)



```
partial interface HTMLBRElement {
  attribute DOMString clear;
};
```

The **clear** IDL attribute of the `<br>` element must [reflect](#) the content attribute of the same name.



```
partial interface HTMLTableCaptionElement {
  attribute DOMString align;
};
```

The **align** IDL attribute of the `<caption>` element must [reflect](#) the content attribute of the same name.



```
partial interface HTMLTableColElement {
  attribute DOMString align;
  attribute DOMString ch;
  attribute DOMString chOff;
  attribute DOMString vAlign;
  attribute DOMString width;
};
```

The **align** and **width** IDL attributes of the `<col>` element must [reflect](#) the respective content attributes of the same name.

The **ch** IDL attribute of the `<col>` element must [reflect](#) the element's `char` content attribute.

The **chOff** IDL attribute of the `<col>` element must [reflect](#) the element's `charoff` content attribute.

The **vAlign** IDL attribute of the `<col>` element must [reflect](#) the element's `valign` content attribute.



User agents must treat `<dir>` elements in a manner equivalent to `ul` elements in terms of semantics and for purposes of rendering.

The `<dir>` element must implement the `HTMLDirectoryElement` interface.

```
interface HTMLDirectoryElement : HTMLElement {
  attribute boolean compact;
};
```

The **compact** IDL attribute of the `<dir>` element must [reflect](#) the content attribute of the same name.



```
partial interface HTMLDivElement {
  attribute DOMString align;
};
```

The **align** IDL attribute of the `<div>` element must [reflect](#) the content attribute of the same name.



```
partial interface HTMLDListElement {
  attribute boolean compact;
};
```

The **compact** IDL attribute of the `<dl>` element must [reflect](#) the content attribute of the same name.



```
partial interface HTMLEmbedElement {
  attribute DOMString align;
  attribute DOMString name;
};
```

The **name** and **align** IDL attributes of the `<embed>` element must [reflect](#) the respective content attributes of the same name.



The `<font>` element must implement the `HTMLFontElement` interface.

```
interface HTMLFontElement : HTMLElement {
  [TreatNullAs=EmptyString] attribute DOMString color;
  attribute DOMString face;
  attribute DOMString size;
};
```

The **color**, **face**, and **size** IDL attributes of the `<font>` element must [reflect](#) the respective content attributes of the same name.



```
partial interface HTMLHeadingElement {
  attribute DOMString align;
};
```

The **align** IDL attribute of the `<h1>`–`<h6>` elements must [reflect](#) the content attribute of the same name.



NOTE:

The **profile** IDL attribute on `<head>` elements (with the `HTMLHeadElement` interface) is intentionally omitted. Unless so required by [another applicable specification](#), implementations would therefore not support this attribute. (It is mentioned here as it was defined in a previous version of the DOM specifications.)



```
partial interface HTMLHRElement {
  attribute DOMString align;
  attribute DOMString color;
  attribute boolean noShade;
  attribute DOMString size;
  attribute DOMString width;
};
```

The **align**, **color**, **size**, and **width** IDL attributes of the `<hr>` element must [reflect](#) the respective content attributes of the same name.

The **noShade** IDL attribute of the `<hr>` element must [reflect](#) the element's `noshade` content attribute.



```
partial interface HTMLHtmlElement {
  attribute DOMString version;
};
```

The **version** IDL attribute of the `<html>` element must [reflect](#) the content attribute of the same name.



```
partial interface HTMLIFrameElement {
  attribute DOMString align;
  attribute DOMString scrolling;
  attribute DOMString frameBorder;

  [TreatNullAs=EmptyString] attribute DOMString marginHeight;
  [TreatNullAs=EmptyString] attribute DOMString marginWidth;
};
```

The **align** and **scrolling** IDL attributes of the `<iframe>` element must [reflect](#) the respective content attributes of the same name.

The **frameBorder** IDL attribute of the `<iframe>` element must [reflect](#) the element's [frameborder](#) content attribute.

The **marginHeight** IDL attribute of the `<iframe>` element must [reflect](#) the element's [marginheight](#) content attribute.

The **marginWidth** IDL attribute of the `<iframe>` element must [reflect](#) the element's [marginwidth](#) content attribute.



```
partial interface HTMLImageElement {
  attribute DOMString name;
  attribute DOMString lowsrc;
  attribute DOMString align;
  attribute unsigned long hspace;
  attribute unsigned long vspace;

  [TreatNullAs=EmptyString] attribute DOMString border;
};
```

The **name**, **align**, **border**, **hspace**, and **vspace** IDL attributes of the `<img>` element must [reflect](#) the respective content attributes of the same name.

The **lowsrc** IDL attribute of the `<img>` element must [reflect](#) the element's [lowsrc](#) content attribute, which for the purposes of reflection is defined as containing a [URL](#).



```
partial interface HTMLInputElement {
  attribute DOMString align;
  attribute DOMString useMap;
};
```

The **align** IDL attribute of the `<input>` element must [reflect](#) the content attribute of the same name.

The **useMap** IDL attribute of the `<input>` element must [reflect](#) the element's [usemap](#) content attribute.



```
partial interface HTMLLegendElement {
  attribute DOMString align;
};
```

The **align** IDL attribute of the `<legend>` element must [reflect](#) the content attribute of the same name.



```
partial interface HTMLLIElement {
  attribute DOMString type;
};
```

The **type** IDL attribute of the `<li>` element must [reflect](#) the content attribute of the same name.



```
partial interface HTMLLinkElement {
  attribute DOMString charset;
  attribute DOMString target;
};
```

The **charset** and **target** IDL attributes of the `<link>` element must [reflect](#) the respective content attributes of the same name.



User agents must treat `<listing>` elements in a manner equivalent to pre elements in terms of semantics and for purposes of rendering.



```
partial interface HTMLMenuElement {
  attribute boolean compact;
};
```

The **compact** IDL attribute of the `<menu>` element must [reflect](#) the content attribute of the same name.



```
partial interface HTMLMetaElement {
  attribute DOMString scheme;
};
```

User agents may treat the [scheme](#) content attribute on the `<meta>` element as an extension of the element's [name](#) content attribute when processing a `<meta>` element with a [name](#) attribute whose value is one that the user agent recognizes as supporting the [scheme](#) attribute.

User agents are encouraged to ignore the [scheme](#) attribute and instead process the value given to the metadata name as if it had been specified for each expected value of the [scheme](#) attribute.

EXAMPLE 666

For example, if the user agent acts on [<meta>](#) elements with [name](#) attributes having the value "eGMS.subject.keyword", and knows that the [scheme](#) attribute is used with this metadata name, then it could take the [scheme](#) attribute into account, acting as if it was an extension of the [name](#) attribute. Thus the following two [<meta>](#) elements could be treated as two elements giving values for two different metadata names, one consisting of a combination of "eGMS.subject.keyword" and "LGCL", and the other consisting of a combination of "eGMS.subject.keyword" and "ORLY":

```
<!-- this markup is invalid -->
<meta name="eGMS.subject.keyword" scheme="LGCL" content="Abandoned vehicles">
<meta name="eGMS.subject.keyword" scheme="ORLY" content="Mah car: kthxbye">
```

The suggested processing of this markup, however, would be equivalent to the following:

```
<meta name="eGMS.subject.keyword" content="Abandoned vehicles">
<meta name="eGMS.subject.keyword" content="Mah car: kthxbye">
```

The **scheme** IDL attribute of the [<meta>](#) element must [reflect](#) the content attribute of the same name.



```
partial interface HTMLObjectElement {
  attribute DOMString align;
  attribute DOMString archive;
  attribute DOMString code;
  attribute boolean declare;
  attribute unsigned long hspace;
  attribute DOMString standby;
  attribute unsigned long vspace;
  attribute DOMString codeBase;
  attribute DOMString codeType;
  attribute DOMString useMap;

  [TreatNullAs=EmptyString] attribute DOMString border;
};
```

The **align**, **archive**, **border**, **code**, **declare**, **hspace**, **standby**, and **vspace** IDL attributes of the [<object>](#) element must [reflect](#) the respective content attributes of the same name.

The **codeBase** IDL attribute of the [<object>](#) element must [reflect](#) the element's [codebase](#) content attribute, which for the purposes of reflection is defined as containing a [URL](#).

The **codeType** IDL attribute of the [<object>](#) element must [reflect](#) the element's [codetype](#) content attribute.

The **useMap** IDL attribute of the [<object>](#) element must [reflect](#) the element's [usemap](#) content attribute.



```
partial interface HTMLListElement {
  attribute boolean compact;
};
```

The **compact** IDL attribute of the [](#) element must [reflect](#) the content attribute of the same name.



```
partial interface HTMLParagraphElement {  
    attribute DOMString align;  
};
```

The **align** IDL attribute of the `<p>` element must [reflect](#) the content attribute of the same name.



```
partial interface HTMLParamElement {  
    attribute DOMString type;  
    attribute DOMString valueType;  
};
```

The **type** IDL attribute of the `<param>` element must [reflect](#) the content attribute of the same name.

The **valueType** IDL attribute of the `<param>` element must [reflect](#) the element's [valueType](#) content attribute.



User agents must treat `<plaintext>` elements in a manner equivalent to `pre` elements in terms of semantics and for purposes of rendering. (The parser has special behavior for this element, though.)



```
partial interface HTMLPreElement {  
    attribute long width;  
};
```

The **width** IDL attribute of the `<pre>` element must [reflect](#) the content attribute of the same name.



```
partial interface HTMLScriptElement {  
    attribute DOMString event;  
    attribute DOMString htmlFor;  
};
```

The **event** IDL attribute of the `<script>` element must reflect the element's [event](#) content attribute.

The **htmlFor** IDL attribute of the `<script>` element must reflect the element's [for](#) content attribute.



```
partial interface HTMLTableElement {
  attribute DOMString align;
  attribute DOMString border;
  attribute DOMString frame;
  attribute DOMString rules;
  attribute DOMString summary;
  attribute DOMString width;

  [TreatNullAs=EmptyString] attribute DOMString bgColor;
  [TreatNullAs=EmptyString] attribute DOMString cellPadding;
  [TreatNullAs=EmptyString] attribute DOMString cellSpacing;
};
```

The **align**, **border**, **frame**, **summary**, **rules**, and **width**, IDL attributes of the `<table>` element must [reflect](#) the respective content attributes of the same name.

The **bgColor** IDL attribute of the `<table>` element must [reflect](#) the element's `bgcolor` content attribute.

The **cellPadding** IDL attribute of the `<table>` element must [reflect](#) the element's `cellpadding` content attribute.

The **cellSpacing** IDL attribute of the `<table>` element must [reflect](#) the element's `cellspacing` content attribute.



```
partial interface HTMLTableSectionElement {
  attribute DOMString align;
  attribute DOMString ch;
  attribute DOMString chOff;
  attribute DOMString vAlign;
};
```

The **align** IDL attribute of the `<tbody>`, `<thead>`, and `<tfoot>` elements must [reflect](#) the content attribute of the same name.

The **ch** IDL attribute of the `<tbody>`, `<thead>`, and `<tfoot>` elements must [reflect](#) the elements' `char` content attributes.

The **chOff** IDL attribute of the `<tbody>`, `<thead>`, and `<tfoot>` elements must [reflect](#) the elements' `charoff` content attributes.

The **vAlign** IDL attribute of the `<tbody>`, `<thead>`, and `<tfoot>` element must [reflect](#) the elements' `valign` content attributes.



```
partial interface HTMLTableCellElement {
  attribute DOMString align;
  attribute DOMString axis;
  attribute DOMString height;
  attribute DOMString width;

  attribute DOMString ch;
  attribute DOMString chOff;
  attribute boolean nowrap;
  attribute DOMString vAlign;

  [TreatNullAs=EmptyString] attribute DOMString bgColor;
};
```

The **align**, **axis**, **height**, and **width** IDL attributes of the `<td>` and `<th>` elements must [reflect](#) the respective content attributes of the same name.

The **ch** IDL attribute of the `<td>` and `<th>` elements must [reflect](#) the elements' `char` content attributes.

The **chOff** IDL attribute of the `<td>` and `<th>` elements must [reflect](#) the elements' [charoff](#) content attributes.

The **nowrap** IDL attribute of the `<td>` and `<th>` elements must [reflect](#) the elements' [nowrap](#) content attributes.

The **vAlign** IDL attribute of the `<td>` and `<th>` element must [reflect](#) the elements' [valign](#) content attributes.

The **bgColor** IDL attribute of the `<td>` and `<th>` elements must [reflect](#) the elements' [bgcolor](#) content attributes.



```
partial interface HTMLTableCellElement {
    attribute DOMString abbr;
};
```

The **abbr** IDL attribute of the `<td>` element must [reflect](#) the respective content attributes of the same name.



```
partial interface HTMLTableRowElement {
    attribute DOMString align;
    attribute DOMString ch;
    attribute DOMString chOff;
    attribute DOMString vAlign;

    [TreatNullAs=EmptyString] attribute DOMString bgColor;
};
```

The **align** IDL attribute of the `<tr>` element must [reflect](#) the content attribute of the same name.

The **ch** IDL attribute of the `<tr>` element must [reflect](#) the element's [char](#) content attribute.

The **chOff** IDL attribute of the `<tr>` element must [reflect](#) the element's [charoff](#) content attribute.

The **vAlign** IDL attribute of the `<tr>` element must [reflect](#) the element's [valign](#) content attribute.

The **bgColor** IDL attribute of the `<tr>` element must [reflect](#) the element's [bgcolor](#) content attribute.



```
partial interface HTMLListElement {
    attribute boolean compact;
    attribute DOMString type;
};
```

The **compact** and **type** IDL attributes of the `<ul>` element must [reflect](#) the respective content attributes of the same name.



User agents must treat `<xmp>` elements in a manner equivalent to `<pre>` elements in terms of semantics and for purposes of rendering. (The parser has special behavior for this element though.)



The `<blink>`, `<bgsound>`, `<isindex>`, `<multicol>`, `<nextid>`, and `<spacer>` elements must use the [HTMLUnknownElement](#) interface.



```
partial interface Document {
  [TreatNullAs=EmptyString] attribute DOMString fgColor;
  [TreatNullAs=EmptyString] attribute DOMString linkColor;
  [TreatNullAs=EmptyString] attribute DOMString vlinkColor;
  [TreatNullAs=EmptyString] attribute DOMString alinkColor;
  [TreatNullAs=EmptyString] attribute DOMString bgColor;

  [SameObject] readonly attribute HTMLCollection anchors;
  [SameObject] readonly attribute HTMLCollection applets;

  void clear();
  void captureEvents();
  void releaseEvents();

  [SameObject] readonly attribute HTMLAllCollection all;
};
```

The attributes of the [Document](#) object listed in the first column of the following table must [reflect](#) the content attribute on the `<body>` element with the name given in the corresponding cell in the second column on the same row, if the `<body>` element is a `<body>` element (as opposed to a `<frameset>` element). When there is no `<body>` element or if it is a `<frameset>` element, the attributes must instead return the empty string on getting and do nothing on setting.

IDL attribute	Content attribute
fgColor	text
linkColor	link
vlinkColor	vlink
alinkColor	alink
bgColor	bgcolor



The **anchors** attribute must return an [HTMLCollection](#) rooted at the [Document](#) node, whose filter matches only `<a>` elements with [name](#) attributes.

The **applets** attribute must return an [HTMLCollection](#) rooted at the [Document](#) node, whose filter matches only `<applet>` elements.

The **clear()**, **captureEvents()**, and **releaseEvents()** methods must do nothing.



The **all** attribute must return an [HTMLAllCollection](#) rooted at the [Document](#) node, whose filter matches all elements.

The object returned for **all** has several unusual behaviors:

- The user agent must act as if the `ToBoolean()` operator in JavaScript converts the object returned for **all** to the false value.
- The user agent must act as if, for the purposes of the `==` and `!=` operators in JavaScript, the object returned for **all** compares as equal to the undefined and null values. (Comparisons using the `===` operator, and comparisons to other values such as strings or objects, are unaffected.)
- The user agent must act such that the `typeof` operator in JavaScript returns the string `undefined` when applied to the object returned for **all**.

NOTE:

These requirements are a [willful violation](#) of the JavaScript specification current at the time of writing (ECMAScript edition 6). The JavaScript specification requires that the `ToBoolean()` operator convert all objects to the true value, and does not have provisions for objects acting as if they were undefined for the purposes of certain operators. This violation is motivated by a desire for compatibility with two classes of legacy content: one that uses the presence of `document.all` as a way to detect legacy user agents, and one that only supports those legacy user agents and uses the `document.all` object without testing for its presence first. [\[ECMA-262\]](#)



```
partial interface Window {
    void captureEvents();
    void releaseEvents();

    [Replaceable, SameObject] readonly attribute External external;
};
```

The `captureEvents()` and `releaseEvents()` methods must do nothing.

The `external` attribute of the [Window](#) interface must return an instance of the [External](#) interface:

```
[NoInterfaceObject]
interface External {
    void AddSearchProvider();
    void IsSearchProviderInstalled();
};
```

The `AddSearchProvider()` and `IsSearchProviderInstalled()` methods must do nothing.

§ 12. IANA considerations

§ 12.1. text/html

This registration is for community review and will be submitted to the IESG for review, approval, and registration with IANA.

Type name:

text

Subtype name:

html

Required parameters:

No required parameters

Optional parameters:

charset

The charset parameter may be provided to specify the [document's character encoding](#), overriding any [character encoding declarations](#) in the document other than a Byte Order Mark (BOM). The parameter's value must be one of the [labels](#) of the [character encoding](#) used to serialize the file. [\[ENCODING\]](#)

Encoding considerations:

8bit (see the section on [character encoding declarations](#))

Security considerations:

Entire novels have been written about the security considerations that apply to HTML documents. Many are listed in this document, to which the reader is referred for more details. Some general concerns bear mentioning here, however:

HTML is scripted language, and has a large number of APIs (some of which are described in this document). Script can expose the user to potential risks of information leakage, credential leakage, cross-site scripting attacks, cross-site request forgeries, and a host of other problems. While the designs in this specification are intended to be safe if implemented correctly, a full implementation is a massive undertaking and, as with any software, user agents are likely to have security bugs.

Even without scripting, there are specific features in HTML which, for historical reasons, are required for broad compatibility with legacy content but that expose the user to unfortunate security problems. In particular, the `<img>` element can be used in conjunction with some other features as a way to effect a port scan from the user's location on the Internet. This can expose local network topologies that the attacker would otherwise not be able to determine.

HTML relies on a compartmentalization scheme sometimes known as the *same-origin policy*. An [origin](#) in most cases consists of all the pages served from the same host, on the same port, using the same protocol.

It is critical, therefore, to ensure that any untrusted content that forms part of a site be hosted on a different [origin](#) than any sensitive content on that site. Untrusted content can easily spoof any other page on the same origin, read data from that origin, cause scripts in that origin to execute, submit forms to and from that origin even if they are protected from cross-site request forgery attacks by unique tokens, and make use of any third-party resources exposed to or rights granted to that origin.

Interoperability considerations:

Rules for processing both conforming and non-conforming content are defined in this specification.

Published specification:

This document is the relevant specification. Labeling a resource with the [text/html](#) type asserts that the resource is an [HTML document](#) using [the HTML syntax](#).

Applications that use this media type:

Web browsers, tools for processing Web content, HTML authoring tools, search engines, validators.

Additional information:**Magic number(s):**

No sequence of bytes can uniquely identify an HTML document. More information on detecting HTML documents is available in the MIME Sniffing specification. [\[MIMESNIFF\]](#)

File extension(s):

"html" and "htm" are commonly, but certainly not exclusively, used as the extension for HTML documents.

Macintosh file type code(s):

TEXT

Person & email address to contact for further information:

Ian Hickson <ian@hixie.ch>

Intended usage:

Common

Restrictions on usage:

No restrictions apply.

Author:

Ian Hickson <ian@hixie.ch>

Change controller:

W3C

Fragment identifiers used with [text/html](#) resources either refer to [the indicated part of the document](#) or provide state information for in-page scripts.

§ 12.2. [multipart/x-mixed-replace](#)

This registration is for community review and will be submitted to the IESG for review, approval, and registration with IANA.

Type name:

multipart

Subtype name:

x-mixed-replace

Required parameters:

- boundary (defined in RFC2046) [\[RFC2046\]](#)

Optional parameters:

No optional parameters.

Encoding considerations:

binary

Security considerations:

Subresources of a `multipart/x-mixed-replace` resource can be of any type, including types with non-trivial security implications such as [text/html](#).

Interoperability considerations:

None.

Published specification:

This specification describes processing rules for Web browsers. Conformance requirements for generating resources with this type are the same as for `multipart/mixed`. [\[RFC2046\]](#)

Applications that use this media type:

This type is intended to be used in resources generated by Web servers, for consumption by Web browsers.

Additional information:**Magic number(s):**

No sequence of bytes can uniquely identify a `multipart/x-mixed-replace` resource.

File extension(s):

No specific file extensions are recommended for this type.

Macintosh file type code(s):

No specific Macintosh file type codes are recommended for this type.

Person & email address to contact for further information:

Ian Hickson <ian@hixie.ch>

Intended usage:

Common

Restrictions on usage:

No restrictions apply.

Author:

Ian Hickson <ian@hixie.ch>

Change controller:

W3C

Fragment identifiers used with `multipart/x-mixed-replace` resources apply to each body part as defined by the type used by that body part.

§ [12.3. application/xhtml+xml](#)

This registration is for community review and will be submitted to the IESG for review, approval, and registration with IANA.

Type name:

application

Subtype name:

xhtml+xml

Required parameters:

Same as for application/xml [\[RFC7303\]](#)

Optional parameters:

Same as for application/xml [\[RFC7303\]](#)

Encoding considerations:

Same as for application/xml [\[RFC7303\]](#)

Security considerations:

Same as for application/xml [\[RFC7303\]](#)

Interoperability considerations:

Same as for application/xml [\[RFC7303\]](#)

Published specification:

Labeling a resource with the application/xhtml+xml type asserts that the resource is an XML document that likely has a root element from the [HTML namespace](#). Thus, the relevant specifications are the XML specification, the Namespaces in XML specification, and this specification. [\[XML\]](#) [\[XPTR-XMLNS\]](#)

Applications that use this media type:

Same as for application/xml [\[RFC7303\]](#)

Additional information:**Magic number(s):**

Same as for application/xml [\[RFC7303\]](#)

File extension(s):

"xhtml" and "xht" are sometimes used as extensions for XML resources that have a root element from the [HTML namespace](#).

Macintosh file type code(s):

TEXT

Person & email address to contact for further information:

Ian Hickson <ian@hixie.ch>

Intended usage:

Common

Restrictions on usage:

No restrictions apply.

Author:

Ian Hickson <ian@hixie.ch>

Change controller:

W3C

Fragment identifiers used with application/xhtml+xml resources have the same semantics as with any [XML MIME type](#). [\[RFC7303\]](#)

§ 12.4. web+ scheme prefix

This section describes a convention for use with the IANA URI scheme registry. It does not itself register a specific scheme. [\[RFC7595\]](#)

Scheme name:

Schemes starting with the four characters "web+" followed by one or more letters in the range a-z.

Status:

Permanent

Scheme syntax:

Scheme-specific.

Scheme semantics:

Scheme-specific.

Encoding considerations:

All "web+" schemes should use UTF-8 encodings where relevant.

Applications/protocols that use this scheme name:

Scheme-specific.

Interoperability considerations:

The scheme is expected to be used in the context of Web applications.

Security considerations:

Any Web page is able to register a handler for all "web+" schemes. As such, these schemes must not be used for features intended to be core platform features (e.g., network transfer protocols like HTTP or FTP). Similarly, such schemes must not store confidential information in their URLs, such as usernames, passwords, personal information, or confidential project names.

Contact:

Ian Hickson <ian@hixie.ch>

Change controller:

Ian Hickson <ian@hixie.ch>

References:

Custom scheme and content handlers, HTML Continually Updated Specification: <https://html.spec.whatwg.org/#custom-handlers>

§ Index

§ Terms defined by this specification

", in §4.7.14

1, in §4.4.5

2d

[context for canvas](#), in §4.12.4

[definition of](#), in §4.12.4

a

[attr-value for ol/type](#), in §4.4.5

[\(element\)](#), in §4.5.1

abbr

[\(element\)](#), in §4.5.9

[element-attr for th](#), in §4.9.10

[attribute for HTMLTableHeaderCellElement](#), in §4.9.10

[element-attr for td](#), in §11.2

[attribute for HTMLTableDataCellElement](#), in §11.3.5

abort

[event for media](#), in §4.7.14.15

[event for global](#), in §Unnumbered section

[abort\(\)](#), in §11.3.4.8

[abort a document](#), in §6.7.12

[abort a parser](#), in §8.2.6

[abort a running script](#), in §7.1.3.6

[aborted](#), in §6.7.12

[aborted prematurely](#), in §7.1.3.6

[aborted the running script](#), in §7.1.3.6

[aborting a document](#), in §6.7.12

[aborting the running script](#), in §7.1.3.6

[abort that parser](#), in §8.2.6

[abort the document](#), in §6.7.12

[abort the image request](#), in §4.7.5

[abort the parser](#), in §8.2.6

[abort the script](#), in §7.1.3.6

[about:](#), in §2.2.2

[about:blank](#), in §2.2.2

[about:legacy-compat](#), in §2.5.1

[about:srcdoc](#), in §2.5.1

[about-to-be-notified rejected promises list](#), in §7.1.3.1

[a browsing context is discarded](#), in §6.3.4

accept

[attribute for HTMLInputElement](#), in §4.10.5

[element-attr for input](#), in §4.10.5.1.17

[element-attr for form](#), in §11.2

[acceptCharset](#), in §4.10.3

[accept-charset](#), in §4.10.3

[Access Key](#), in §4.11.6.1

[accessKey](#), in §5.5.3

[accesskey](#), in §5.5.2

[acknowledged](#), in §8.2.4

[acknowledge the token's self-closing flag](#), in §8.2.4

[acronym](#), in §11.2

[Action](#), in §4.11.6.1

[action](#)

[element-attr for form](#), in §4.10.19.6

[definition of](#), in §4.10.19.6

[attribute for HTMLFormElement](#), in §4.10.19.6

[activation behavior](#), in §5.3

[:active](#), in §4.15.2

[activeCues](#), in §4.7.14.11.5

[active document](#), in §6.1

[activeElement](#), in §5.4.6

[active flag](#), in §4.7.14.11.1

[active flag was set when the script started](#), in §4.7.14.11.5

[active frame element](#), in §11.3.3

[active parser](#), in §3.1.2

[active range](#), in §5.6.4

[active sandboxing flag set](#), in §6.5

[actually disabled](#), in §4.14

[actual value](#), in §4.10.15

[add\(\)](#), in §5.7.3.1

[addCue\(cue\)](#), in §4.7.14.11.5

[add\(data\)](#), in §5.7.3.1

[add\(data, type\)](#), in §5.7.3.1

[add\(element\)](#)

[method for HTMLOptionsCollection](#), in §2.7.2.3

[method for HTMLSelectElement](#), in §4.10.7

[add\(element, before\)](#)

[method for HTMLOptionsCollection](#), in §2.7.2.3

[method for HTMLSelectElement](#), in §4.10.7

[additional allowed character](#), in §8.2.4.69

[address](#)

[definition of](#), in §3.1

[\(element\)](#), in §4.3.9

[AddSearchProvider\(\)](#), in §11.3.5

[addTextTrack\(kind, label, language\)](#), in §4.7.14.11.5

[addtrack](#), in §4.7.14.15

[adjusted current node](#), in §8.2.3.2

[adjust foreign attributes](#), in §8.2.5.1

[adjust MathML attributes](#), in §8.2.5.1

[adjust SVG attributes](#), in §8.2.5.1

[administrative level](#), in §4.10.19.8.1

[adoption agency algorithm](#), in §8.2.5.4.7

[a drag data item kind](#), in §5.7.2

[a drag data item type string](#), in §5.7.2

[advance](#), in §4.9.12.1

[advanced to the next child of the table](#), in §4.9.12.1

[advisory information](#), in §3.2.5.2

[a fallback entry](#), in §11.3.4

[affected by a base URL change](#), in §2.2.2

[after after body](#), in §8.2.5.4.22

[after after frameset](#), in §8.2.5.4.23

[after body](#), in §8.2.5.4.19

[after frameset](#), in §8.2.5.4.21

[after head](#), in §8.2.5.4.6

[afterprint](#), in §Unnumbered section

[afterscriptexecute](#), in §Unnumbered section

[a known definite encoding](#), in §8.2.2.1

[a label](#), in §4.7.14.11.1

[a language](#), in §4.7.14.11.1

[alert\(\)](#), in §7.5.1

[alert\(message\)](#), in §7.5.1

[algorithm for assigning header cells](#), in §4.9.12.2

[algorithm for ending a row group](#), in §4.9.12.1

[algorithm for extracting a character encoding from a meta element](#), in §2.6.5

[algorithm for growing downward-growing cells](#), in §4.9.12.1

[algorithm for processing row groups](#), in §4.9.12.1

[algorithm for processing rows](#), in §4.9.12.1

[algorithm to convert a Date object to a string](#), in §4.10.5

[algorithm to convert a number to a string](#), in §4.10.5

[algorithm to convert a string to a Date object](#), in §4.10.5

[algorithm to convert a string to a number](#), in §4.10.5

[align](#)

[element-attr for caption](#), in §11.2

[element-attr for col](#), in §11.2

[element-attr for div](#), in §11.2

[element-attr for embed](#), in §11.2

[element-attr for hr](#), in §11.2

[element-attr for headings](#), in §11.2

[element-attr for iframe](#), in §11.2

[element-attr for input](#), in §11.2

[element-attr for img](#), in §11.2

[element-attr for legend](#), in §11.2

[element-attr for object](#), in §11.2

[element-attr for p](#), in §11.2

[element-attr for table](#), in §11.2

[element-attr for tbody, tthead, tfoot, tablesection](#), in §11.2

[element-attr for td, th, tablecells](#), in §11.2

[element-attr for tr](#), in §11.2

[attribute for HTMLAppletElement](#), in §11.3.1

[attribute for HTMLTableCaptionElement](#), in §11.3.5

[attribute for HTMLTableColElement](#), in §11.3.5

[attribute for HTMLDivElement](#), in §11.3.5

[attribute for HTMLEmbedElement](#), in §11.3.5

[attribute for HTMLHeadingElement](#), in §11.3.5

[attribute for HTMLHRElement](#), in §11.3.5

[attribute for HTMLIFrameElement](#), in §11.3.5

[attribute for HTMLImageElement](#), in §11.3.5

[attribute for HTMLInputElement](#), in §11.3.5

[attribute for HTMLLegendElement](#), in §11.3.5

[attribute for HTMLObjectElement](#), in §11.3.5

[attribute for HTMLParagraphElement](#), in §11.3.5

[attribute for HTMLTableElement](#), in §11.3.5

[attribute for HTMLTableSectionElement](#), in §11.3.5

[attribute for HTMLTableCellElement](#), in §11.3.5

[attribute for HTMLTableRowElement](#), in §11.3.5

[align descendants](#), in §10.2

[alink](#), in §11.2

- [aLink](#), in §11.3.5
- [alinkColor](#), in §11.3.5
- [a list of zero or more cues](#), in §4.7.14.11.1
- [all](#)
 - [value for effectAllowed](#), in §5.7.3
 - [attribute for Document](#), in §11.3.5
- ["all"-named elements](#), in §2.7.2.1
- [allowed keywords and their meanings](#), in §4.8.6
- [allowed to navigate](#), in §6.1.3
- [allowed to show a popup](#), in §6.1.5
- [allowed value step](#), in §4.10.5.3.8
- [allow-forms](#), in §6.5
- [allowfullscreen](#), in §4.7.6
- [allowFullscreen](#), in §4.7.6
- [allow-modals](#), in §6.5
- [allow-pointer-lock](#), in §6.5
- [allow-popups](#), in §6.5
- [allow-popups-to-escape-sandbox](#), in §6.5
- [allow-same-origin](#), in §6.5
- [allow-scripts](#), in §6.5
- [allow-top-navigation](#), in §6.5
- [allowtransparency](#), in §11.2
- [alphanumeric ASCII characters](#), in §2.4.1
- ["already started"](#), in §4.12.1.1
- [alt](#)
 - [element-attr for img](#), in §4.7.5
 - [attribute for HTMLImageElement](#), in §4.7.5
 - [element-attr for area](#), in §4.7.16
 - [attribute for HTMLAreaElement](#), in §4.7.16
 - [attribute for HTMLInputElement](#), in §4.10.5
 - [element-attr for input](#), in §4.10.5.1.19
 - [attribute for HTMLAppletElement](#), in §11.3.1
- [alternate](#)
 - [attr-value for link/type](#), in §4.8.6.1
 - [attr-value for marquee/behavior](#), in §11.3.2
 - [state for marquee/behavior](#), in §11.3.2
- [alternative](#), in §4.7.14.10.1
- [ambiguous ampersand](#), in §8.1.4
- [a mode](#), in §4.7.14.11.1
- [an alternative stylesheet](#), in §4.8.6.10
- [ancestor](#), in §6.1.1
- [ancestor browsing context](#), in §6.1.1
- [ancestorOrigins](#), in §6.6.4
- [ancestor origins array](#), in §6.6.4
- [anchors](#), in §11.3.5
- [an end time](#), in §4.7.14.11.1
- [An entry with persisted user state](#), in §6.6.1
- [a new date object](#), in §2.2.2
- [a new start for session storage](#), in §6.1.5
- [an explicit entry](#), in §11.3.4
- [an identifier](#), in §4.7.14.11.1
- [an iframe srcdoc documents](#), in §4.7.6
- [animation frame callback identifier](#), in §7.8
- [an in-band metadata track dispatch type](#), in §4.7.14.11.1
- [an indicated part of the document](#), in §6.7.9
- [annotates](#), in §4.8.1
- [annotation pairing](#), in §4.5.10
- [anonymous](#), in §2.6.6
- [Anonymous](#), in §2.6.6
- [an overridden reload](#), in §3.1
- [An unstyled document view](#), in §10.9
- [any](#), in §4.8.6.5
- [API base URL](#), in §7.1.3.1
- [APIs](#), in §4.10.5.4
- [API URL character encoding](#), in §7.1.3.1
- [API value](#), in §4.10.11
- [a plausible language](#), in §7.6.1.2
- [appCodeName](#), in §7.6.1.1
- [applet](#), in §11.3.1
- [applets](#), in §11.3.5
- [applicable specification](#), in §2.2.3
- [application cache](#), in §11.3.4
- [ApplicationCache](#), in §11.3.4.8
- [applicationCache](#)
 - [attribute for Window](#), in §11.3.4.8
 - [attribute for SharedWorkerGlobalScope](#), in §11.3.4.8
- [application cache download process](#), in §11.3.4.2
- [application cache group](#), in §11.3.4
- [application cache manifest](#), in §11.3.4
- [application cache mode](#), in §11.3.4
- [application cache selection](#), in §11.3.4.3
- [application cache selection algorithm](#), in §11.3.4.3
- [application-name](#), in §4.2.5.1
- [application/x-www-form-urlencoded](#), in §4.10.19.6
- [application/x-www-form-urlencoded encoding algorithm](#), in §4.10.22.6
- [apply](#), in §4.10.5
- [appName](#), in §7.6.1.1
- [appropriate end tag token](#), in §8.2.4
- [appropriate form encoding algorithm](#), in §4.10.22.3
- [appropriate place for inserting a node](#), in §8.2.5.1
- [appropriate template contents owner document](#), in §4.12.3
- [appVersion](#), in §7.6.1.1
- [archive](#)
 - [element-attr for object](#), in §11.2
 - [attribute for HTMLAppletElement](#), in §11.3.1
 - [attribute for HTMLObjectElement](#), in §11.3.5
- [area](#), in §4.7.16
- [a readiness state](#), in §4.7.14.11.1
- [areas](#), in §4.7.15
- [a registered handler](#), in §7.6.1.3
- [aria-*](#), in §2.2.2
- [aria-checked](#), in §2.2.2
- [aria-describedby](#), in §2.2.2
- [aria-disabled](#), in §2.2.2
- [aria-expanded](#), in §2.2.2
- [aria-hidden](#), in §2.2.2
- [aria-invalid](#), in §2.2.2
- [aria-label](#), in §2.2.2
- [aria-level](#), in §2.2.2
- [aria-multiline](#), in §2.2.2
- [aria-multiselectable](#), in §2.2.2
- [aria-owns](#), in §2.2.2
- [aria-readonly](#), in §2.2.2
- [aria-required](#), in §2.2.2
- [aria-selected](#), in §2.2.2

[aria-sort](#), in §2.2.2

[aria-valuemax](#), in §2.2.2

[aria-valuemin](#), in §2.2.2

[aria-valuenow](#), in §2.2.2

[art directed](#), in §4.7.1

[art direction](#), in §4.7.1

[article](#), in §4.3.2

[as a download](#), in §4.8.5

[ASCII case-insensitive](#), in §2.3

[ASCII-compatible encoding](#), in §2.1.6

[ASCII digits](#), in §2.4.1

[ASCII hex digits](#), in §2.4.1

[ASCII serialization of an origin](#), in §6.4

[a serialization of the bitmap as a file](#), in §4.12.4.2

[a serialization of the canvas element's bitmap as a file](#), in §4.12.4.2

[as hints for the rendering](#), in §10.2

[aside](#), in §4.3.5

[ask for a reset](#), in §4.10.7

[asks for a reset](#), in §4.10.7

[as part of an attribute](#), in §8.2.4.41

[assign\(\)](#), in §6.6.4

[assigned access key](#), in §5.5.3

[assigned context menu](#), in §4.11.5.2

[assigned media provider object](#), in §4.7.14.2

[assign\(url\)](#), in §6.6.4

[associated](#), in §4.10.18.3

[associated content-type headers](#), in §2.6.4

[associated content-type metadata](#), in §2.6.4

[associated inert template document](#), in §4.12.3

[associate section](#), in §4.3.10.1

[a start time](#), in §4.7.14.11.1

[as that element in the specific scope](#), in §8.2.3.2

[a style sheet that is blocking scripts](#), in §4.2.7

[async](#)

[element-attr for script](#), in §4.12.1

[attribute for HTMLScriptElement](#), in §4.12.1

[atob\(atob\)](#), in §7.2

[attribute](#), in §8.1.2.3

[attribute name](#), in §8.1.2.3

[attribute names](#), in §8.1.2.3

[attributes](#), in §8.1.2.3

[attribute serialized name](#), in §8.3

[Attributes for form submission](#), in §4.10.19.6

[attribute value](#), in §8.1.2.3

[attribute values](#), in §8.1.2.3

[a type change is signalled](#), in §4.10.5

[A type that the user agent knows it cannot render](#), in §4.7.14.3

[audio](#), in §4.7.11

[Audio\(src\)](#), in §4.7.11

[AudioTrack](#), in §4.7.14.10.1

[AudioTrack.enabled](#), in §4.7.14.10.1

[AudioTrack.id](#), in §4.7.14.10.1

[AudioTrack.kind](#), in §4.7.14.10.1

[AudioTrack.label](#), in §4.7.14.10.1

[AudioTrack.language](#), in §4.7.14.10.1

[AudioTrackList](#), in §4.7.14.10.1

[AudioTrackList.getTrackById\(id\)](#), in §4.7.14.10.1

[AudioTrackList.length](#), in §4.7.14.10.1

[audioTracks](#), in §4.7.14.10

[author](#)

[definition of](#), in §4.2.5.1

[attr-value for link/type](#), in §4.8.6.2

["auto"](#), in §6.6.2

[auto](#)

[attr-value for global/dir](#), in §3.2.5.6

[state for dir](#), in §3.2.5.6

[value for HTMLMediaElement/preload](#), in §4.7.14.5

[state for scope](#), in §4.9.10

[value for scrollRestorationMode](#), in §6.6.1

[enum-value for ScrollRestoration](#), in §6.6.2

[autocomplete](#)

[attribute for HTMLFormElement](#), in §4.10.3

[element-attr for autocompleteelements, form, input, select, textarea](#), in §4.10.19.8.1

[attribute for HTMLInputElement, HTMLSelectElement, HTMLTextAreaElement](#), in §4.10.19.8.2

[autofill](#), in §4.10.19.8

[autofill anchor mantle](#), in §4.10.19.8.1

[Autofill detail tokens](#), in §4.10.19.8.1

[autofill expectation mantle](#), in §4.10.19.8.1

[autofill field](#), in §4.10.19.8.1

[autofill field name](#), in §4.10.19.8.2

[autofill hint set](#), in §4.10.19.8.2

[autofill mechanism](#), in §4.10.19.8

[autofill scope](#), in §4.10.19.8.2

[autofills form controls](#), in §4.10.19.8.2

[autofocus](#)

[element-attr for formelements, input, button, keygen, select, textarea](#), in §4.10.19.6.1

[attribute for HTMLInputElement, HTMLButtonElement, HTMLSelectElement, HTMLTextAreaElement, HTMLKeygenElement](#), in §4.10.19.6.1

[Automatic](#), in §4.7.14.5

[autoplay](#)

[element-attr for media](#), in §4.7.14.7

[attribute for HTMLMediaElement](#), in §4.7.14.7

[autoplaying flag](#), in §4.7.14.5

[auxiliary browsing context](#), in §6.1.2

[auxiliary browsing contexts](#), in §6.1.2

[Available](#), in §4.7.5

[available](#)

[state for img](#), in §4.7.5

[definition of](#), in §4.10.5.1.19

[await a stable state](#), in §7.1.4.2

[a WebSocket message has been received](#), in §2.2.2

[axis](#)

[element-attr for td](#), in §11.2

[attribute for HTMLTableCellElement](#), in §11.3.5

[b](#), in §4.5.23[back\(\)](#), in §6.6.2[background](#)[element-attr for body, table, thead, tbody, tfoot, tr, td, th, common](#), in §11.2[attribute for HTMLBodyElement](#), in §11.3.5[badInput](#), in §4.10.21.3[BarProp](#), in §6.3.6[barred from constraint validation](#), in §4.10.21.1[barring it from constraint validation](#), in §4.10.21.1[base](#), in §4.2.3[basefont](#), in §11.2[base URL change steps](#), in §2.2.2[bdi](#), in §4.5.26[bdo](#), in §4.5.27[before head](#), in §8.2.5.4.3[before html](#), in §8.2.5.4.2[beforeprint](#), in §Unnumbered section[beforescriptexecute](#), in §Unnumbered section[beforeunload](#), in §Unnumbered section[BeforeUnloadEvent](#), in §6.7.11.1[behavior](#)[element-attr for marquee](#), in §11.3.2
[attribute for HTMLMarqueeElement](#), in §11.3.2[being activated](#), in §4.15.2[being actively pointed at](#), in §4.15.2[being rendered](#), in §10.1[being unloaded](#), in §6.7.11[being used as relevant canvas fallback content](#), in §4.12.4[best floating-point number](#), in §2.4.4.3[bgcolor](#)[element-attr for body](#), in §11.2
[element-attr for table](#), in §11.2
[element-attr for td, th, tablecells](#), in §11.2
[element-attr for tr](#), in §11.2[bgColor](#)[attribute for HTMLMarqueeElement](#), in §11.3.2[attribute for HTMLBodyElement](#), in §11.3.5[attribute for HTMLTableElement](#), in §11.3.5[attribute for HTMLTableCellElement](#), in §11.3.5[attribute for HTMLTableRowElement](#), in §11.3.5[attribute for Document](#), in §11.3.5[bgsound](#), in §11.2[bidirectional-algorithm formatting character ranges](#), in §3.2.6.1[big](#), in §11.2[Big5](#), in §8.2.2.3[billing](#), in §4.10.19.8.1[blank](#), in §10.3.10[blink](#), in §11.2[blob:](#), in §2.2.2[blocked by a modal dialog](#), in §5.2[blocked by the modal dialog](#), in §5.2[blocked media element](#), in §4.7.14.8[blocked-on-parser](#), in §4.7.14.11.1[blockquote](#), in §4.4.4[blur](#), in §Unnumbered section[blur\(\)](#), in §5.4.6[body](#)[attribute for Document](#), in §3.1.3
(element), in §4.3.1[bookmark](#), in §4.8.6.3[boolean attribute](#), in §2.4.2[boolean attributes](#), in §2.4.2[border](#)[element-attr for HTMLTableElement](#), in §4.9.1[element-attr for input](#), in §11.2[element-attr for img](#), in §11.2[element-attr for object](#), in §11.2[element-attr for table](#), in §11.2[attribute for HTMLImageElement](#), in §11.3.5[attribute for HTMLObjectElement](#), in §11.3.5[attribute for HTMLTableElement](#), in §11.3.5[bordercolor](#), in §11.2[bottommargin](#), in §11.2[bounce](#), in §11.3.2[br](#), in §4.5.29[broken](#), in §4.7.5[Broken](#), in §4.7.5[browsing context](#), in §6.1[browsing context container](#), in §6.1.1[browsing context name](#), in §6.1.5[browsing contexts](#), in §6.1[browsing context scope origin](#), in §6.1.3[btoa\(btoa\)](#), in §7.2[buffered](#), in §4.7.14.5[build a menu construct](#), in §4.11.3[build and show](#), in §4.11.3[build and show a menu](#), in §4.11.3[Button](#)[element-state for input](#), in §4.10.5.1.21[element-state for button/type](#), in §4.10.6[button](#)[attr-value for input/type](#), in §4.10.5
(element), in §4.10.6[attr-value for button/type](#), in §4.10.6[buttons](#), in §4.10.2[cache attempt](#), in §11.3.4.2[cache failure steps](#), in §11.3.4.2[cache host](#), in §11.3.4[cache mode flag](#), in §11.3.4[cache selection](#), in §11.3.4.3[calling scripts](#), in §7.1.3.4[can be focused](#), in §5.4.3[cancelAnimationFrame\(\)](#), in §7.8[cancelAnimationFrame\(handle\)](#), in §7.8[canceled activation steps](#), in §5.3[candidate for constraint validation](#), in §4.10.21.1[candidates for constraint validation](#), in §4.10.21.1[canplay](#), in §4.7.14.15[canplaythrough](#), in §4.7.14.15[canPlayType\(type\)](#), in §4.7.14.3[canvas](#), in §4.12.4[canvas blob serialization task source](#), in §4.12.4[canvas context mode](#), in §4.12.4

caption

[attribute for HTMLTableElement](#), in §4.9.1

[\(element\)](#), in §4.9.2

captions

[attr-value for track/kind](#), in §4.7.13

[attr-value for commonTrack/kind](#), in §4.7.14.10.1

[definition of](#), in §4.7.14.11.1

[enum-value for TextTrackKind](#), in §4.7.14.11.5

[Captions](#), in §4.7.13

["captions"](#), in §4.7.14.11.5

captureEvents()

[method for Document](#), in §11.3.5

[method for Window](#), in §11.3.5

[case-sensitive](#), in §2.3

[Categories](#), in §3.2.3

[causes the opener attribute to remain null](#), in §6.1.5

[CDATA sections](#), in §8.1.5

[cell](#), in §4.9.12

[cellIndex](#), in §4.9.11

[cellpadding](#), in §11.2

[cellPadding](#), in §11.3.5

cells

[attribute for HTMLTableRowElement](#), in §4.9.8

[definition of](#), in §4.9.12

[cellspacing](#), in §11.2

[cellSpacing](#), in §11.3.5

[center](#), in §11.2

ch

[attribute for HTMLTableColElement](#), in §11.3.5

[attribute for HTMLTableSectionElement](#), in §11.3.5

[attribute for HTMLTableCellElement](#), in §11.3.5

[attribute for HTMLTableRowElement](#), in §11.3.5

challenge

[element-attr for keygen](#), in §4.10.12

[attribute for HTMLKeygenElement](#), in §4.10.12

change

[definition of](#), in §2.1.3

[event for MediaList](#), in §4.7.14.15

[event for input](#), in §4.10.5.5

[event for global](#), in §Unnumbered section

[change the encoding](#), in §8.2.2.4

["chapters"](#), in §4.7.14.11.5

[Chapters](#), in §4.7.13

chapters

[attr-value for track/kind](#), in §4.7.13

[definition of](#), in §4.7.14.11.1

[enum-value for TextTrackKind](#), in §4.7.14.11.5

char

[element-attr for col](#), in §11.2

[element-attr for tbody, thead, tfoot, tablesection](#), in §11.2

[element-attr for td, th, tablecells](#), in §11.2

[element-attr for tr](#), in §11.2

[character](#), in §2.1.6

[character encoding](#), in §2.1.6

[character encoding declaration](#), in §4.2.5.5

[character height](#), in §4.10.11

[character references](#), in §8.1.4

[character width](#), in §4.10.11

charoff

[element-attr for col](#), in §11.2

[element-attr for tbody, thead, tfoot, tablesection](#), in §11.2

[element-attr for td, th, tablecells](#), in §11.2

[element-attr for tr](#), in §11.2

[charset](#), in §4.10.19.1

charset

[element-attr for meta](#), in §4.2.5

[element-attr for script](#), in §4.12.1

[attribute for HTMLScriptElement](#), in §4.12.1

[element-attr for a](#), in §11.2

[element-attr for link](#), in §11.2

[attribute for HTMLAnchorElement](#), in §11.3.5

[attribute for HTMLLinkElement](#), in §11.3.5

Checkbox

[element-state for input](#), in §4.10.5.1.15

[state for menuitem](#), in §4.11.4

checkbox

[attr-value for input/type](#), in §4.10.5

[attr-value for menuitem/type](#), in §4.11.4

checked

[element-attr for input](#), in §4.10.5

[attribute for HTMLInputElement](#), in §4.10.5.4

[element-attr for menuitem](#), in §4.11.4

[attribute for HTMLMenuItemElement](#), in §4.11.4

[:checked](#), in §4.15.2

[checkedness](#), in §4.10.18.1

[check if we can run script](#), in §7.1.3.4

[CHECKING](#), in §11.3.4.8

checkValidity()

[method for HTMLFormElement](#), in §4.10.3

[method for HTMLObjectElement, HTMLInputElement, HTMLButtonElement, HTMLSelectElement, HTMLTextAreaElement, HTMLKeygenElement, HTMLOutputElement, HTMLFieldSetElement](#), in §4.10.21.3

[child browsing context](#), in §6.1.1

[child browsing context name property set](#), in §6.3.3

[child browsing contexts](#), in §6.1.1

[child text content](#), in §2.1.3

chOff

[attribute for HTMLTableColElement](#), in §11.3.5

[attribute for HTMLTableSectionElement](#), in §11.3.5

[attribute for HTMLTableCellElement](#), in §11.3.5

[attribute for HTMLTableRowElement](#), in §11.3.5

[circ](#), in §4.7.16

[circle](#), in §4.7.16

[circle state](#), in §4.7.16

[Circle state](#), in §4.7.16

cite

[element-attr for blockquote](#), in §4.4.4

[\(element\)](#), in §4.5.6

[element-attr for q](#), in §4.5.7

[element-attr for edits](#), in §4.6.3

[attribute for HTMLModElement](#), in §4.6.3

[class](#), in §3.2.5.7

[classic script](#), in §7.1.3.1

[classid](#), in §11.2

[clean up after running script](#), in §7.1.3.4

clear

[element-attr for br](#), in §11.2

[attribute for HTMLBRElement](#), in §11.3.5

clear()

[method for DataTransferItemList](#), in §5.7.3.1

[method for Document](#), in §11.3.5

[clearData\(\)](#), in §5.7.3

[clearData\(format\)](#), in §5.7.3

[clearInterval\(\)](#), in §7.4

[clearInterval\(handle\)](#), in §7.4

[clear the list of active formatting elements up to the last marker](#), in §8.2.3.3

[clear the stack back to a table body context](#), in §8.2.5.4.13

[clear the stack back to a table context](#), in §8.2.5.4.9

[clear the stack back to a table row context](#), in §8.2.5.4.14

[clearTimeout\(\)](#), in §7.4

[clearTimeout\(handle\)](#), in §7.4

[click\(\)](#), in §5.3

[Clone](#), in §2.9.1

[Cloneable objects](#), in §2.9.1

close()

[method for Window](#), in §6.3.1

[method for Document](#), in §7.3.2

[close a browsing context](#), in §6.3.5

[close a p element](#), in §8.2.5.4.7

[closed](#), in §6.3.1

[close the cell](#), in §8.2.5.4.15

[close the WebSocket connection](#), in §2.2.2

[closing misnested formatting elements](#), in §8.2.5.4.7

code

[\(element\)](#), in §4.5.17

[attribute for MediaError](#), in §4.7.14.1

[element-attr for object](#), in §11.2

[attribute for HTMLAppletElement](#), in §11.3.1

[attribute for HTMLObjectElement](#), in §11.3.5

codeBase

[attribute for HTMLAppletElement](#), in §11.3.1

[attribute for HTMLObjectElement](#), in §11.3.5

[codebase](#), in §11.2

[codetype](#), in §11.2

[codeType](#), in §11.3.5

[code unit](#), in §2.1.6

[code-unit length](#), in §2.1.6

col

[\(element\)](#), in §4.9.4

[attr-value for scope](#), in §4.9.10

colgroup

[\(element\)](#), in §4.9.3

[attr-value for scope](#), in §4.9.10

[colgroup group](#), in §4.9.10

[collect a sequence of characters](#), in §2.4.1

[colno](#), in §7.1.3.8.2

[Color](#), in §4.10.5.1.14

color

[attr-value for input/type](#), in §4.10.5

[element-attr for hr](#), in §11.2

[attribute for HTMLFontElement](#), in §11.3.5

[attribute for HTMLHRElement](#), in §11.3.5

cols

[element-attr for textarea](#), in §4.10.11

[attribute for HTMLTextAreaElement](#), in §4.10.11

[element-attr for frameset](#), in §11.3.3

[colspan](#), in §4.9.11

[colSpan](#), in §4.9.11

column

[state for scope](#), in §4.9.10

[definition of](#), in §4.9.12

[column group](#), in §4.9.12

[column group header](#), in §4.9.12.2

[column groups](#), in §4.9.12

[column header](#), in §4.9.12.2

[columns](#), in §4.9.12

[Command](#), in §4.11.4

command

[attr-value for menuitem/type](#), in §4.11.4

[dfn for menuitem](#), in §4.11.6.1

[comment](#), in §8.1.6

[commentary](#), in §4.7.14.10.1

[comments](#), in §8.1.6

[commit an automatic annotation](#), in §4.5.13

[commit an automatic base](#), in §4.5.10

[commit a ruby segment](#), in §4.5.10

[commit current annotations](#), in §4.5.10

[commit the base range](#), in §4.5.10

compact

[element-attr for dl](#), in §11.2

[element-attr for menu](#), in §11.2

[element-attr for ol](#), in §11.2

[element-attr for ul](#), in §11.2

[attribute for HTMLDirectoryElement](#), in §11.3.5

[attribute for HTMLDListElement](#), in §11.3.5

[attribute for HTMLMenuElement](#), in §11.3.5

[attribute for HTMLListElement](#), in §11.3.5

[attribute for HTMLULListElement](#), in §11.3.5

[comparing origins](#), in §6.4

[compatibility caseless](#), in §2.3

[compiled pattern regular expression](#), in §4.10.5.3.6

["complete"](#), in §3.1.1

complete

[enum-value for DocumentReadyState](#), in §3.1.1

[attribute for HTMLImageElement](#), in §4.7.5

[Completely available](#), in §4.7.5

[completely available](#), in §4.7.5

[completely loaded](#), in §8.2.6

[completeness flag](#), in §11.3.4

[compound microtasks](#), in §7.1.4.2

[compound microtask subtask](#), in §7.1.4.2

[computed MIME type](#), in §2.6.4

[computed type of a resource](#), in §2.6.4

[computed type of the resource](#), in §2.6.4

[conent category](#), in §3.2.4.2

[confidence](#), in §8.2.2

[confirm\(\)](#), in §7.5.1

[confirm\(message\)](#), in §7.5.1

[conforming document](#), in §2.2.1

[conforming documents](#), in §2.2.1

[constraint validation API](#), in §4.10.21.3

- [Constructing the form data set](#), in §4.10.22.4
- [construct the form data set](#), in §4.10.22.4
- [consume a character reference](#), in §8.2.4.69
- [consumed](#), in §8.2.2.5
- [container frame element](#), in §10.3.2
- content
 - [element-attr for meta](#), in §4.2.5
 - [attribute for HTMLMetaElement](#), in §4.2.5
 - [attribute for HTMLTemplateElement](#), in §4.12.3
- [content attribute](#), in §3.2.3
- [content attributes](#), in §3.2.3
- [content categories](#), in §3.2.4.2
- contentDocument
 - [attribute for HTMLIFrameElement](#), in §4.7.6
 - [attribute for HTMLFrameElement](#), in §11.3.3
- [contentEditable](#), in §5.6.1
- [contenteditable](#), in §5.6.1
- [content-language](#), in §4.2.5.3
- [Content model](#), in §3.2.3
- [Content security policy state](#), in §4.2.5.3
- content-type
 - [definition of](#), in §2.6.4
 - [state for http-equiv](#), in §4.2.5.3
- [content-type metadata](#), in §2.6.4
- contentWindow
 - [attribute for HTMLIFrameElement](#), in §4.7.6
 - [attribute for HTMLFrameElement](#), in §11.3.3
- context
 - [attr-value for menu/type](#), in §4.11.3
 - [definition of](#), in §8.4
- [contextMenu](#), in §4.11.5.2
- [context menu](#), in §4.11.5
- contextmenu
 - [element-attr for global](#), in §4.11.5.1
 - [event for global](#), in §Unnumbered section
- [context mode](#), in §4.12.4
- [Contexts in which this element can be used](#), in §3.2.3
- [control](#), in §4.10.4
- [control characters](#), in §2.4.1
- [control group](#), in §5.4.2
- [control group owner](#), in §5.4.2
- [control group owner object](#), in §5.4.2
- [control group owner objects](#), in §5.4.2
- controls
 - [attribute for Document](#), in §3.2.1
 - [element-attr for mediaelements](#), video, audio, in §4.7.14.12
 - [attribute for HTMLMediaElement](#), in §4.7.14.12
- [control's data](#), in §4.10.19.8.2
- [controls in the user interface that is exposed to the user](#), in §4.7.14.12
- [convert a list of dimensions to a list of pixel values](#), in §10.6
- [converting a character width to pixels](#), in §10.5.4
- [Converting a string to ASCII lower-case](#), in §2.3
- [Converting a string to ASCII upper-case](#), in §2.3
- [cookie](#), in §3.1.2
- [cookie-averse](#), in §3.1.2
- [cookieEnabled](#), in §7.6.1.4
- [cookies set during the server's opening handshake](#), in §2.2.2
- [cookie-string](#), in §2.2.2
- [coordinate](#), in §4.10.5.1.19
- coords
 - [element-attr for area](#), in §4.7.16
 - [attribute for HTMLAreaElement](#), in §4.7.16
 - [element-attr for a](#), in §11.2
 - [attribute for HTMLAnchorElement](#), in §11.3.5
- copy
 - [value for dropEffect](#), in §5.7.3
 - [value for effectAllowed](#), in §5.7.3
 - [value for drag](#), in §5.7.5
 - [definition of](#), in §5.7.8
 - [event for global](#), in §Unnumbered section
- [copyLink](#), in §5.7.3
- [copyMove](#), in §5.7.3
- [cors-same-origin](#), in §6.4
- [CORS settings attribute](#), in §2.6.6
- [create a classic script](#), in §7.1.3.3
- [create a drag data store](#), in §5.7.2
- [create an element for a token](#), in §8.2.5.1
- [create an element for the token](#), in §8.2.5.1
- [create a new browsing context](#), in §6.1
- [create a potential-CORS request](#), in §2.6.1
- [create a script](#), in §7.1.3.3
- [createCaption\(\)](#), in §4.9.1
- [createImageBitmap\(\)](#), in §7.7
- [createImageBitmap\(image\)](#), in §7.7
- [createImageBitmap\(image, sx, sy, sw, sh\)](#), in §7.7
- [createTBody\(\)](#), in §4.9.1
- [createTFoot\(\)](#), in §4.9.1
- [createTHead\(\)](#), in §4.9.1
- [creating a classic script](#), in §7.1.3.3
- [creating a new browsing context](#), in §6.1
- [creating a potential-cors request](#), in §2.6.1
- [creating scripts](#), in §7.1.3.3
- [creation URL](#), in §7.1.3.1
- [creator base URL](#), in §6.1
- [creator browsing context](#), in §6.1
- [creator origin](#), in §6.1
- [creator URL](#), in §6.1
- [critical subresource](#), in §2.1.1
- [critical subresources](#), in §2.1.1
- [crop bitmap data to the source rectangle](#), in §7.7
- [cropped to the source rectangle](#), in §7.7
- [cross-boundary event parent](#), in §2.2.2
- crossorigin
 - [element-attr for link](#), in §4.2.4
 - [element-attr for img](#), in §4.7.5
 - [element-attr for media](#), in §4.7.14.2
 - [element-attr for script](#), in §4.12.1

crossOrigin

- [attribute for HTMLLinkElement](#), in §4.2.4
- [attribute for HTMLImageElement](#), in §4.7.5
- [attribute for HTMLMediaElement](#), in §4.7.14.2
- [attribute for HTMLScriptElement](#), in §4.12.1

cross-origin, in §6.4**CrossOriginFunctionWrapper**, in §6.2.3.3.2**CrossOriginGet**, in §6.2.3.4**CrossOriginGetOwnPropertyHelper**, in §6.2.3.3**CrossOriginOwnPropertyKeys**, in §6.2.3.6**CrossOriginProperties**, in §6.2.3.1**CrossOriginPropertyDescriptor**, in §6.2.3.3.1**CrossOriginPropertyDescriptorMap**, in §6.2.2**CrossOriginSet**, in §6.2.3.5**cross-origin wrapper function**, in §6.2.3.3.2**CSP list**, in §3.1.1**CSS properties**, in §2.1**CSS property**, in §2.1**cue**, in §4.7.14.11.1**cuechange**, in §4.7.14.15**cues**

- [definition of](#), in §4.7.14.11.1
- [attribute for TextTrack](#), in §4.7.14.11.5

Current, in §7.1.3.5**current document readiness**, in §3.1.2**current drag operation**, in §5.7.5**current entry**, in §6.6.1**current entry of the joint session history**, in §6.6.2**current global object**, in §7.1.3.5.3**current input character**, in §8.2.2.5**currently focused area of a top-level browsing context**, in §5.4.2**currently focused area of the top-level browsing context**, in §5.4.2**currently relevant menu element**, in §4.11.3**currently running task**, in §7.1.4.1**current node**, in §8.2.3.2**current pixel density**, in §4.7.5**current playback position**, in §4.7.14.6**current position**, in §4.7.14.6**current request**, in §4.7.5**currentScript**, in §3.1.3**current settings object**, in §7.1.3.5.3**currentSrc**

- [attribute for HTMLImageElement](#), in §4.7.5
- [attribute for HTMLMediaElement](#), in §4.7.14.2

current target element, in §5.7.5**current template insertion mode**, in §8.2.3.1**currentTime**, in §4.7.14.6**current URL**, in §4.7.5**current value**, in §4.10.14**custom data attribute**, in §3.2.5.9**customError**, in §4.10.21.3**custom validity error message**, in §4.10.21.1**cut**, in §Unnumbered section**data-**, in §3.2.5.9**data:**, in §2.2.2**data**

- [\(element\)](#), in §4.5.15
- [element-attr for object](#), in §4.7.8
- [attribute for DataCue](#), in §4.7.14.11.6

data-*, in §3.2.5.9**data block**, in §4.12.1**data blocks**, in §4.12.1**DataCue**, in §4.7.14.11.6**DataCue(startTime, endTime, data)**, in §4.7.14.11.6**datafld**, in §11.2**dataformatas**, in §11.2**datalist**, in §4.10.8**datapagesize**, in §11.2**datasrc**, in §11.2**dataTransfer**, in §5.7.4**DataTransfer**, in §5.7.3**DataTransferItem**, in §5.7.3.2**DataTransferItemList**, in §5.7.3.1**data url**, in §2.2.2**data: url**, in §2.2.2**date**

- [dfn for dates](#), in §2.4.5.2
- [attr-value for input/type](#), in §4.10.5

Date, in §4.10.5.1.7**date object**, in §2.2.2**dates**, in §2.4.5.2**datetime**

- [element-attr for time](#), in §4.5.16
- [element-attr for edits](#), in §4.6.3

dateTime

- [attribute for HTMLTimeElement](#), in §4.5.16
- [attribute for HTMLModElement](#), in §4.6.3

datetime-local, in §4.10.5**datetime value**, in §4.5.16**dd**, in §4.4.10**decimal**, in §4.4.5**declare**

- [element-attr for object](#), in §11.2
- [attribute for HTMLObjectElement](#), in §11.3.5

dedicated media source failure steps, in §4.7.14.5**Default**, in §4.10.19.7**:default**, in §4.15.2**default**

- [element-attr for track](#), in §4.7.13
- [attribute for HTMLTrackElement](#), in §4.7.13
- [attr-value for area/shape](#), in §4.7.16
- [mode for input](#), in §4.10.5.4
- [mode for output](#), in §4.10.13
- [element-attr for menuitem](#), in §4.11.4
- [attribute for HTMLMenuItemElement](#), in §4.11.4

default behavior, in §5.6.5**default button**, in §4.10.22.2**defaultChecked**, in §4.10.5**default commands**, in §4.11.3**default maximum**, in §4.10.5.3.7**default minimum**, in §4.10.5.3.7**defaultMuted**, in §4.7.14.12**default object size**, in §2.2.2**default/on**, in §4.10.5.4**defaultPlaybackRate**, in §4.7.14.8**default playback start position**, in §4.7.14.6

[DefaultProperties](#), in §6.6.4

[defaultSelected](#), in §4.10.10

[default state](#), in §4.7.16

[Default state](#), in §4.7.16

[default step](#), in §4.10.5.3.8

[default step base](#), in §4.10.5.3.8

[default-style](#), in §4.2.5.3

[default value](#)

[dfn for range](#), in §4.10.5.1.13

[dfn for output](#), in §4.10.13

[defaultValue](#)

[attribute for HTMLInputElement](#), in §4.10.5

[attribute for HTMLTextAreaElement](#), in §4.10.11

[attribute for HTMLOutputElement](#), in §4.10.13

[defaultView](#), in §6.3

[defer](#)

[element-attr for script](#), in §4.12.1

[attribute for HTMLScriptElement](#), in §4.12.1

[defines a command](#), in §4.11.6.1

[defines the term](#), in §4.5.8

[defining term](#), in §4.5.8

[del](#), in §4.6.2

[delaying load events mode](#), in §6.1.1

[delaying the load event](#), in §8.2.6

[delaying-the-load-event flag](#), in §4.7.14.5

[delays the load event](#), in §8.2.6

[delay the load event](#), in §8.2.6

[delete an existing named property](#), in §2.7.3

[deleteCaption\(\)](#), in §4.9.1

[deleteCell\(index\)](#), in §4.9.8

[deleteRow\(index\)](#)

[method for HTMLTableElement](#), in §4.9.1

[method for HTMLTableSectionElement](#), in §4.9.5

[deleteTFoot\(\)](#), in §4.9.1

[deleteTHead\(\)](#), in §4.9.1

[delete the selection](#), in §5.6.4

[density-corrected intrinsic width and height](#), in §4.7.5

[dereferencing a javascript: url](#), in §6.7.1

[derived from country in some cases](#), in §4.10.19.8.2

[described above](#), in §4.7.5.1.16

[description](#)

[definition of](#), in §4.2.5.1

[attribute for Plugin](#), in §7.6.1.5

[attribute for MimeType](#), in §7.6.1.5

[descriptions](#)

[attr-value for track/kind](#), in §4.7.13

[attr-value for commonTrack/kind](#), in §4.7.14.10.1

[definition of](#), in §4.7.14.11.1

[enum-value for TextTrackKind](#), in §4.7.14.11.5

["descriptions"](#), in §4.7.14.11.5

[Descriptions](#), in §4.7.13

[designated pop-up menu](#), in §4.10.6

[designMode](#)

[attribute for Document](#), in §3.1.1

[definition of](#), in §5.6.2

[Detached](#), in §2.9.2

[detach from a media element](#), in §2.2.2

[details](#), in §4.11.1

[details notification task steps](#), in §4.11.1

[determining the type of the resource](#), in §4.2.4

[device-pixel-ratio](#), in §4.7.1

[dfn](#), in §4.5.8

[dialog arguments](#), in §7.5.3

[dialogArguments](#), in §7.5.3

[dialog arguments' origin](#), in §7.5.3

[did-perform-automatic-track-selection](#), in §4.7.14.11.1

[Dimension attributes](#), in §4.7.20

[dir](#)

[attribute for Document](#), in §3.1.1

[element-attr for global](#), in §3.2.5.6

[\(element\)](#), in §11.2

[direction](#)

[element-attr for marquee](#), in §11.3.2

[attribute for HTMLMarqueeElement](#), in §11.3.2

[directionality](#), in §3.2.5.6

[directionality-capable attributes](#), in §3.2.5.6

[directionality of an attribute](#), in §3.2.5.6

[directionality of the attribute](#), in §3.2.5.6

[direction of playback](#), in §4.7.14.8

[directly reachable browsing contexts](#), in §6.1.4

[:dir\(ltr\)](#), in §4.15.2

[dirname](#), in §4.10.19.2

[dirName](#)

[attribute for HTMLInputElement](#), in §4.10.5

[attribute for HTMLTextAreaElement](#), in §4.10.11

[:dir\(rtl\)](#), in §4.15.2

[dirtiness](#), in §4.10.10

[dirty checkedness](#), in §4.10.5

[dirty checkedness flag](#), in §4.10.5

[dirty value flag](#)

[dfn for input](#), in §4.10.5

[dfn for textarea](#), in §4.10.11

[:disabled](#), in §4.15.2

["disabled"](#), in §4.7.14.11.5

[disabled](#)

[mode for track](#), in §4.7.14.11.1

[enum-value for TextTrackMode](#), in §4.7.14.11.5

[enum for TextTrackMode](#), in §4.7.14.11.5

[element-attr for optgroup](#), in §4.10.9

[attribute for HTMLOptGroupElement](#), in §4.10.9

[element-attr for option](#), in §4.10.10

[attribute for HTMLOptionElement](#), in §4.10.10

[element-attr for fieldset](#), in §4.10.16

[attribute for HTMLFieldSetElement](#), in §4.10.16

[element-attr for disabledfor elements, input, button, keygen, select, textarea](#), in §4.10.19.5

[attribute for HTMLInputElement](#),

[HTMLButtonElement](#),

[HTMLSelectElement](#),

[HTMLTextAreaElement](#),

[HTMLKeygenElement](#), in §4.10.19.5

[element-attr for menuitem](#), in §4.11.4

[attribute for](#)

[HTMLMenuItemElement](#), in §4.11.4

[disabled fieldset](#), in §4.10.16

[Disabled State](#), in §4.11.6.1

[disabling](#), in §4.14

[discard](#), in §6.3.4

[discard a document](#), in §6.3.4

[discarded](#), in §6.3.4

[discard the document](#), in §6.3.4

[disown its opener](#), in §6.1.2.1

[dispatch](#), in §2.1.4

[dispatched](#), in §2.1.4

[dispatching](#), in §2.1.4

[displayed](#), in §2.1

[display size](#), in §4.10.7

[display state](#), in §4.7.14.11.1

[display the inline content](#), in §6.7.8

[div](#), in §4.4.14

[dl](#), in §4.4.8

[DOCTYPE](#), in §8.1.1

[DOCTYPE legacy string](#), in §8.1.1

[document](#)

[definition of](#), in §2.1

[attribute for Window](#), in §6.3

[Document](#), in §3.1.1

[DocumentAndElementEventHandlers](#)

, in §7.1.5.2.1

[document associated with a window](#), in §6.1

[document base URL](#), in §2.5.1

[document family](#), in §6.1.1

[documents](#), in §2.1

[does not apply](#), in §4.10.5

[doesn't apply](#), in §4.10.5

[doesn't necessarily have to affect](#), in §6.6.3

[domain](#), in §6.4.1

[DOM anchor](#), in §5.4.2

[DOMContentLoaded](#), in §Unnumbered section

[DOMElementMap](#), in §2.7.4

[dom event dispatch logic](#), in §2.1.4

[DOM interface](#), in §3.2.3

[DOM manipulation task source](#), in §7.1.4.3

[DOMStringMap](#), in §2.7.3

[do not apply](#), in §4.10.5

[do not set](#), in §3.2.7.1

[do not support scripting](#), in §2.2.1

[don't apply](#), in §4.10.5

[down](#)

[attr-value for marquee/direction](#), in §10.5.11

[state for marquee](#), in §11.3.2

[download](#)

[attribute for HTMLAnchorElement](#), in §4.5.1

[attribute for HTMLAreaElement](#), in §4.7.16

[element-attr for a, area, links](#), in §4.8.2

[definition of](#), in §4.8.5

[download hyperlinks](#), in §4.8.5

[DOWNLOADING](#), in §11.3.4.8

[downloads a hyperlink](#), in §4.8.5

[download the hyperlink](#), in §4.8.5

[drag](#), in §5.7.6

[drag-and-drop events](#), in §5.7.6

[drag data item kind](#), in §5.7.2

[drag data item type string](#), in §5.7.2

[drag data item type strings](#), in §5.7.2

[drag data store](#), in §5.7.2

[drag data store allowed effects state](#), in §5.7.2

[drag data store bitmap](#), in §5.7.2

[drag data store default feedback](#), in §5.7.2

[drag data store hot spot coordinate](#), in §5.7.2

[drag data store item list](#), in §5.7.2

[drag data store mode](#), in §5.7.2

[dragend](#), in §5.7.6

[dragenter](#), in §5.7.6

[DragEvent](#), in §5.7.4

[DragEvent\(type\)](#), in §5.7.4

[DragEvent\(type, eventInitDict\)](#), in §5.7.4

[dragexit](#), in §5.7.6

[draggable](#)

[element-attr for global](#), in §5.7.7

[attribute for HTMLElement](#), in §5.7.7

[dragleave](#), in §5.7.6

[dragover](#), in §5.7.6

[dragstart](#), in §5.7.6

[drop](#), in §5.7.6

[dropEffect](#), in §5.7.3

[dropzone](#)

[element-attr for global](#), in §5.7.8

[attribute for HTMLElement](#), in §5.7.8

[dropzone processing steps](#), in §5.7.8

[dt](#), in §4.4.9

[duration](#)

[definition of](#), in §2.4.5.9

[attribute for HTMLMediaElement](#), in §4.7.14.6

[durationchange](#), in §4.7.14.15

[duration time component](#), in §2.4.5.9

[duration time component scale](#), in §2.4.5.9

[during form submission](#), in §4.10.22.4

[dynamic markup insertion](#), in §7.3

[earliest possible position](#), in §4.7.14.6

[earliest possible position when the script started](#), in §4.7.14.11.5

[editable](#), in §5.6.4

[editing host](#), in §5.6.4

[effectAllowed](#), in §5.7.3

[effective domain](#), in §6.4

[effective media volume](#), in §4.7.14.12

[effective playback rate](#), in §4.7.14.8

[ElementContentEditable](#), in §5.6.1

[element has the focus](#), in §4.15.2

[elements](#)

[attribute for HTMLFormElement](#), in §4.10.3

[attribute for HTMLFieldSetElement](#), in §4.10.16

[elements with default margins](#), in §10.3.10

[element type](#), in §2.1.2

[element with default margins](#), in §10.3.10

[em](#), in §4.5.2

[E-mail](#), in §4.10.19.7

[email](#)

[attr-value for input/type](#), in §4.10.5

[element-state for input](#), in §4.10.5.1.5

[value for form/inputmode](#), in §4.10.19.7

[e-mail](#), in §4.10.5.1.5

[embed](#), in §4.7.7

[embedded content](#), in §3.2.4.2.6

[Embedding custom non-visible data](#), in §3.2.5.9

[embeds](#), in §3.1.3

[embed task source](#), in §4.7.7

[emptied](#), in §4.7.14.15

[empty](#), in §2.1.3

[empty cell](#), in §4.9.12.2

[enabled](#), in §4.7.14.10.1

[:enabled](#), in §4.15.2

[enabledPlugin](#), in §7.6.1.5

[encoding](#), in §4.10.19.6

[encoding declaration state](#), in §4.2.5.3

[encoding labels](#), in §2.1.6

[encoding name](#), in §2.1.6

[encoding sniffing algorithm](#), in §8.2.2.2

[enctype](#)

- [element-attr for form](#), in §4.10.19.6
- [definition of](#), in §4.10.19.6
- [attribute for HTMLFormElement](#), in §4.10.19.6

[end](#)

- [enum-value for SelectionMode](#), in §4.10.20
- [enum for SelectionMode](#), in §4.10.20

["end"](#), in §4.10.20

[ended](#)

- [definition of](#), in §4.7.14.8
- [event for media](#), in §4.7.14.15

[ended playback](#), in §4.7.14.8

[end\(index\)](#), in §4.7.14.13

[end tag](#), in §8.1.2.2

[end tags](#), in §8.1.2.2

[endTime](#), in §4.7.14.11.5

[enter](#), in §4.7.14.15

[entrance counter](#), in §7.1.3.5.1

[Entry](#), in §7.1.3.5

[entry execution context](#), in §7.1.3.5.1

[entry global object](#), in §7.1.3.5.1

[entry Realm](#), in §7.1.3.5.1

[entry settings object](#), in §7.1.3.5.1

[entry update](#), in §6.7.1

[enumerated attributes](#), in §2.4.3

[environment settings object](#), in §7.1.3.1

[error](#)

- [attribute for HTMLMediaElement](#), in §4.7.14.1
- [event for media](#), in §4.7.14.15
- [event for source](#), in §4.7.14.15
- [event for track](#), in §4.7.14.15
- [attribute for ErrorEvent](#), in §7.1.3.8.2
- [event for global](#), in §Unnumbered section

[ERROR](#), in §4.7.13

[ErrorEvent](#), in §7.1.3.8.2

[ErrorEvent\(type\)](#), in §7.1.3.8.2

[ErrorEvent\(type, eventInitDict\)](#), in §7.1.3.8.2

[error occurs during reading of the object](#), in §2.2.2

[escapable raw text](#), in §8.1.2

[escapable raw text elements](#), in §8.1.2

[Escaping a string](#), in §8.3

[establish a WebSocket connection](#), in §2.2.2

[establishing the media timeline](#), in §4.7.14.6

[establish the media timeline](#), in §4.7.14.6

[EUC-KR](#), in §8.2.2.3

[event](#)

- [element-attr for script](#), in §11.2
- [attribute for HTMLScriptElement](#), in §11.3.5

[event dispatching](#), in §2.1.4

[event handler](#), in §7.1.5.1

[EventHandler](#), in §7.1.5.1

[event handler content attribute](#), in §7.1.5.1

[event handler content attributes](#), in §7.1.5.1

[event handler event type](#), in §7.1.5.1

[event handler IDL attribute](#), in §7.1.5.1

[event handler IDL attributes](#), in §7.1.5.1

[EventHandlerNonNull](#), in §7.1.5.1

[event handlers](#), in §7.1.5.1

[event loop](#), in §7.1.4.1

[event loops](#), in §7.1.4.1

[event parent](#), in §2.2.2

[exceptions enabled flag](#), in §6.7.1

[execCommand\(\)](#), in §5.6.4

[execCommand\(commandId\)](#), in §3.1.1

[execCommand\(commandId, showUI\)](#), in §3.1.1

[execCommand\(commandId, showUI, value\)](#), in §3.1.1

[execute](#), in §4.12.1.1

[execute a compound microtask subtask](#), in §7.1.4.2

[execute a script block](#), in §4.12.1.1

[execute the script block](#), in §4.12.1.1

[existing](#), in §2.7.4

[exit](#), in §4.7.14.15

[explicit content-type metadata](#), in §2.6.4

[explicit entries](#), in §11.3.4

[explicit "EOF" character](#), in §8.2.2.5

[explicitly going back or forwards in the session history](#), in §6.6.2

[explicitly supported](#), in §7.6.1.5

[explicitly supported JSON type](#), in §6.7.1

[explicitly supported XML type](#), in §6.7.1

[explicitly supports](#), in §7.6.1.5

[explicit section](#), in §11.3.4.1

[expose a user interface to the user](#), in §4.7.14.12

[exposed](#), in §3.1.3

[exposes a user interface to the user](#), in §4.7.14.12

[exposing a user interface](#), in §4.7.14.12

[exposing a user interface to the user](#), in §4.7.14.12

[expressly inert](#), in §5.4.2

[extension](#), in §4.8.5

[extensions in use](#), in §2.2.2

[extensions to the predefined set of link type](#), in §4.8.6.13

[extensions to the predefined set of link types](#), in §4.8.6.13

[Extensions to the predefined set of pragma directives](#), in §4.2.5.4

[External](#), in §11.3.5

[external](#), in §11.3.5

- [external resource](#), in §4.8.1
- [external resource link](#), in §4.8.1
- [external resources](#), in §4.8.1
- [face](#), in §11.3.5
- [facets](#), in §4.11.6.1
- [failed](#), in §4.7.14.11.1
- [failed to load](#), in §4.7.14.11.1
- [fail the WebSocket connection](#), in §2.2.2
- [fallback base URL](#), in §2.5.1
- [fallback content](#), in §3.2.4.2.6
- [fallback entries](#), in §11.3.4
- [fallback namespaces](#), in §11.3.4
- [fallback section](#), in §11.3.4.1
- [false-by-default](#), in §5.6.5
- [familiar with](#), in §6.1.3
- [fast](#), in §11.3.4
- [fastSeek\(time\)](#), in §4.7.14.9
- [feed the parser](#), in §9.2
- [fetch a classic script](#), in §7.1.3.2
- [fetch a classic worker script](#), in §7.1.3.2
- [fetching a classic script](#), in §7.1.3.2
- [Fetching scripts](#), in §7.1.3.2
- [fgColor](#), in §11.3.5
- [fieldset](#), in §4.10.16
- [figcaption](#), in §4.4.12
- [figure](#), in §4.4.11
- [file](#), in §4.10.5
- [filename](#)
 - [mode for input](#), in §4.10.5.4
 - [attribute for ErrorEvent](#), in §7.1.3.8.2
 - [attribute for Plugin](#), in §7.6.1.5
- [files](#)
 - [attribute for HTMLInputElement](#), in §4.10.5.4
 - [attribute for DataTransfer](#), in §5.7.3
- [file upload](#), in §4.10.5.1.17
- [file upload controls](#), in §4.10.5.1.17
- [finish](#), in §11.3.2
- [finishes](#), in §8.2.6
- [fire](#), in §2.1.4
- [fire a click event](#), in §7.1.5.3
- [fire a DND event](#), in §5.7.4
- [fire a focus event](#), in §5.4.4
- [fire a progress event](#), in §4.7.5
- [fire a progress event or simple event](#), in §4.7.5
- [fire a simple event](#), in §2.1.4
- [fire a synthetic mouse event named contextmenu](#), in §7.1.5.3
- [fired](#), in §2.1.4
- [fired unload](#), in §6.7.11
- [fires](#), in §2.1.4
- [fires a simple event](#), in §2.1.4
- [firing](#), in §2.1.4
- [firing a click event](#), in §7.1.5.3
- [firing a simple event](#), in §7.1.5.3
- [firing a simple event named e](#), in §7.1.5.3
- [firing a synthetic mouse event named click](#), in §7.1.5.3
- [Firing a synthetic mouse event named e](#), in §7.1.5.3
- [floating date and time](#), in §2.4.5.5
- [flow content](#), in §3.2.4.2.2
- [focus\(\)](#), in §5.4.6
- [focus](#), in §Unnumbered section
- [:focus](#), in §4.15.2
- [focusable](#), in §5.4.2
- [focusable area](#), in §5.4.2
- [focus chain](#), in §5.4.2
- [focused](#), in §5.4.2
- [focused area](#), in §5.4.2
- [focused area of a control group](#), in §5.4.2
- [focused area of that focus group](#), in §5.4.2
- [focused area of the control group](#), in §5.4.2
- [focused dialog](#), in §5.4.2
- [Focus fixup rule one](#), in §5.4.4
- [focusing steps](#), in §5.4.4
- [focus update steps](#), in §5.4.4
- [follow hyperlinks](#), in §4.8.4
- [following a hyperlink](#), in §4.8.4
- [following hyperlinks](#), in §4.8.4
- [follows a hyperlink](#), in §4.8.4
- [follow the hyperlink](#), in §4.8.4
- [follow the hyperlinks](#), in §4.8.4
- [font](#), in §11.2
- [footer](#), in §4.3.8
- [for](#)
 - [element-attr for label](#), in §4.10.4
 - [element-attr for output](#), in §4.10.13
 - [element-attr for script](#), in §11.2
- [forced sandboxing flag set](#), in §6.5
- [force-quirks flag](#), in §8.2.4
- [forces content into a unique origin](#), in §6.4
- [forceSpellCheck\(\)](#), in §5.6.5
- [foreign](#), in §11.3.4
- [Foreign elements](#), in §8.1.2
- [forget the media element's media-resource-specific tracks](#), in §4.7.14.5
- [form](#)
 - [\(element\)](#), in §4.10.3
 - [element-attr for formelements, object, label, input, button, select, textarea, keygen, output, fieldset](#), in §4.10.18.3
 - [attribute for FormIDLAttribute, HTMLObjectElement, HTMLLabelElement, HTMLInputElement, HTMLButtonElement, HTMLSelectElement, HTMLOptionElement, HTMLTextAreaElement, HTMLKeygenElement, HTMLOutputElement, HTMLFieldSetElement, HTMLLegendElement](#), in §4.10.18.3
- [formation](#), in §4.10.19.6
- [formAction](#), in §4.10.19.6
- [form-associated](#), in §4.10.2
- [form-associated elements](#), in §4.10.2
- [formatting](#), in §8.2.3.2
- [formatting element tags](#), in §8.2.3.2
- [form control maxlength attribute](#), in §4.10.19.3
- [form control minlength attribute](#), in §4.10.19.4
- [form element pointer](#), in §8.2.3.4
- [formEncType](#), in §4.10.19.6
- [formenctype](#), in §4.10.19.6
- [formMethod](#), in §4.10.19.6
- [formmethod](#), in §4.10.19.6
- [formNoValidate](#), in §4.10.19.6
- [formnovalidate](#), in §4.10.19.6
- [form owner](#), in §4.10.18.3
- [Forms](#), in §4.10

- [forms](#), in §3.1.3
- [form submission](#), in §4.10.22
- [form submission algorithm](#), in §4.10.22.3
- [form submissions](#), in §4.10.22
- [formTarget](#), in §4.10.19.6
- [formtarget](#), in §4.10.19.6
- [for privacy](#), in §1.8
- [forward\(\)](#), in §6.6.2
- [foster parent](#), in §8.2.5.1
- [foster parented](#), in §8.2.5.1
- [foster parenting](#), in §8.2.5.1
- [fragment case](#), in §8.4
- [fragment identifier](#), in §6.7.9
- [fragment identifiers](#), in §6.7.9
- [frame](#)
 - [element-attr for table](#), in §11.2
 - [\(element\)](#), in §11.3.3
 - [attribute for HTMLTableElement](#), in §11.3.5
- [frameborder](#), in §11.2
- [frameBorder](#)
 - [attribute for HTMLFrameElement](#), in §11.3.3
 - [attribute for HTMLIFrameElement](#), in §11.3.5
- [frame border color](#), in §10.6
- [frameElement](#), in §6.1.1.1
- [frames](#), in §6.3
- [frameset](#), in §11.3.3
- [frameset-ok flag](#), in §8.2.3.5
- [framespacing](#), in §11.2
- [from an external file](#), in §4.12.1.1
- [frozen base URL](#), in §4.2.3
- [Full-width Latin](#), in §4.10.19.7
- [full-width-latin](#), in §4.10.19.7
- [fully active](#), in §6.1.1
- [fully decodable](#), in §4.7.5
- [further restrictions](#), in §8.1.2.6
- [gain focus](#), in §5.4.2
- [gb18030](#), in §8.2.2.3
- [generate all implied end tags thoroughly](#), in §8.2.5.3
- [generate implied end tags](#), in §8.2.5.3
- [generator](#), in §4.2.5.1
- [generator-unable-to-provide-required-alt](#), in §4.7.5.1.22
- [generic raw text element parsing algorithm](#), in §8.2.5.2
- [generic RCDATA element parsing algorithm](#), in §8.2.5.2
- [get](#), in §4.10.19.6
- [GET](#), in §4.10.19.6
- [Get action URL](#), in §4.10.22.3
- [get all-indexed](#), in §2.7.2.1
- [get all-named](#), in §2.7.2.1
- [get an attribute](#), in §8.2.2.2
- [getAsFile\(\)](#), in §5.7.3.2
- [getAsString\(callback\)](#), in §5.7.3.2
- [getContext\(contextId, arguments...\)](#), in §4.12.4
- [getContext\(contextId, arguments\)](#), in §4.12.4
- [getCueById\(id\)](#), in §4.7.14.11.5
- [getData\(format\)](#), in §5.7.3
- [getElementsByName\(\)](#), in §3.1.3
- [getElementsByName\(elementName\)](#), in §3.1.1
- [gets reset](#), in §6.7.10
- [getStartDate\(\)](#), in §4.7.14.6
- [get the current value of the event handler](#), in §7.1.5.1
- [getting](#), in §2.1.4
- [Getting an encoding](#), in §2.2.2
- [getting an output encoding](#), in §2.2.2
- [getting the current value of the event handler](#), in §7.1.5.1
- [getTrackById\(id\)](#)
 - [method for AudioTrackList](#), in §4.7.14.10.1
 - [method for VideoTrackList](#), in §4.7.14.10.1
 - [method for TextTrackList](#), in §4.7.14.11.5
- [global aria-* attributes](#), in §3.2.7.3.2
- [Global attributes](#), in §3.2.5
- [global date and time](#), in §2.4.5.7
- [GlobalEventHandlers](#), in §7.1.5.2.1
- [global object](#), in §7.1.3.5
- [global script clean-up jobs list](#), in §7.1.3.4
- [global States and properties](#), in §3.2.7.3.2
- [go\(\)](#), in §6.6.2
- [go\(delta\)](#), in §6.6.2
- [group](#), in §4.9.12
- [Guidelines for exposing cues](#), in §4.7.14.11.4
- [h1](#), in §4.3.6
- [h2](#), in §4.3.6
- [h3](#), in §4.3.6
- [h4](#), in §4.3.6
- [h5](#), in §4.3.6
- [h6](#), in §4.3.6
- [handled](#)
 - [dfn for script](#), in §7.1.3.8
 - [dfn for promise](#), in §7.1.3.9
- [hard](#), in §4.10.11
- [Hard](#), in §4.10.11
- [hardware limitations](#), in §2.2.1
- [has a border](#), in §10.6
- [has an effect](#), in §11.3.4.3
- [has a p element in button scope](#), in §8.2.3.2
- [has a periodic domain](#), in §4.10.5.3.7
- [has a reversed range](#), in §4.10.5.3.7
- [has a style sheet that is blocking scripts](#), in §4.2.7
- [hasFocus\(\)](#), in §5.4.6
- [has focus steps](#), in §5.4.4
- [hash](#)
 - [attribute for HTMLHyperlinkElementUtils](#), in §4.8.3
 - [attribute for Location](#), in §6.6.4
- [hashchange](#), in §Unnumbered section
- [HashChangeEvent](#), in §6.7.10.3
- [HashChangeEvent\(type\)](#), in §6.7.10.3
- [HashChangeEvent\(type, eventInitDict\)](#), in §6.7.10.3
- [@@hasInstance](#), in §2.2.2
- [has no style sheet that is blocking scripts](#), in §4.2.7
- [has range limitations](#), in §4.10.5.3.7
- [has that element in the specific scope](#), in §8.2.3.2
- [have an element in table scope](#), in §8.2.3.2
- [have an element target node in a specific scope](#), in §8.2.3.2

[have an li element in list item scope](#), in §8.2.3.2

[have a particular element in button scope](#), in §8.2.3.2

[have a particular element in list item scope](#), in §8.2.3.2

[have a particular element in select scope](#), in §8.2.3.2

[have a particular element in table scope](#), in §8.2.3.2

[have a p element in button scope](#), in §8.2.3.2

[have a periodic domain](#), in §4.10.5.3.7

[have a reversed range](#), in §4.10.5.3.7

[have a style sheet that is blocking scripts](#), in §4.2.7

[HAVE CURRENT DATA](#), in §4.7.14.7

[HAVE ENOUGH DATA](#), in §4.7.14.7

[HAVE FUTURE DATA](#), in §4.7.14.7

[HAVE METADATA](#), in §4.7.14.7

[HAVE NOTHING](#), in §4.7.14.7

[have range limitations](#), in §4.10.5.3.7

head

[attribute for Document](#), in §3.1.3

[\(element\)](#), in §4.2.1

[head element pointer](#), in §8.2.3.4

[header](#), in §4.3.7

headers

[element-attr for tablecells](#), in §4.9.11

[attribute for HTMLTableCellElement](#), in §4.9.11

[headers to send appropriate cookies](#), in §2.2.2

[heading content](#), in §3.2.4.2.4

[headings](#), in §3.2.4.2.4

height

[attribute for HTMLImageElement](#), in §4.7.5

[attribute for HTMLIFrameElement](#), in §4.7.6

[attribute for HTMLEmbedElement](#), in §4.7.7

[attribute for HTMLObjectElement](#), in §4.7.8

[element-attr for media, img, iframe, embed, object, video, input](#), in §4.7.20

[attribute for media](#), in §4.7.20

[attribute for HTMLInputElement](#), in §4.10.5

[element-attr for canvas](#), in §4.12.4

[attribute for HTMLCanvasElement](#), in §4.12.4

[attribute for ImageBitmap](#), in §7.7

[element-attr for table](#), in §11.2

[element-attr for td, th, tablecells](#), in §11.2

[element-attr for tr](#), in §11.2

[attribute for HTMLAppletElement](#), in §11.3.1

[attribute for HTMLMarqueeElement](#), in §11.3.2

[attribute for HTMLTableCellElement](#), in §11.3.5

[help](#), in §4.8.6.4

hidden

[mode for track](#), in §4.7.14.11.1

[enum-value for TextTrackMode](#), in §4.7.14.11.5

[enum for TextTrackMode](#), in §4.7.14.11.5

[attr-value for input/type](#), in §4.10.5

[definition of](#), in §5.1

[attribute for common](#), in §5.1

[Hidden](#), in §4.10.5.1.1

["hidden"](#), in §4.7.14.11.5

[hidden plugins](#), in §7.6.1.5

[Hidden State](#), in §4.11.6.1

high

[element-attr for meter](#), in §4.10.15

[attribute for HTMLMeterElement](#), in §4.10.15

[high boundary](#), in §4.10.15

[History](#), in §6.6.2

[history](#), in §6.6.1

[history traversal](#), in §6.7.10

[history traversal task source](#), in §7.1.4.3

[home control group](#), in §5.4.5

[home sequential focus navigation order](#), in §5.4.5

[home subtree](#), in §2.1.3

[home subtrees](#), in §2.1.3

[honor user preferences for automatic text track selection](#), in §4.7.14.11.3

host

[attribute for HTMLHyperlinkElementUtils](#), in §4.8.3

[attribute for Location](#), in §6.6.4

hostname

[attribute for HTMLHyperlinkElementUtils](#), in §4.8.3

[attribute for Location](#), in §6.6.4

[:hover](#), in §4.15.2

[hr](#), in §4.4.2

href

[element-attr for base](#), in §4.2.3

[attribute for HTMLBaseElement](#), in §4.2.3

[element-attr for link](#), in §4.2.4

[attribute for HTMLLinkElement](#), in §4.2.4

[element-attr for a, area, links](#), in §4.8.2

[attribute for HTMLHyperlinkElementUtils](#), in §4.8.3

[attribute for Location](#), in §6.6.4

hreflang

[element-attr for link](#), in §4.2.4

[attribute for HTMLLinkElement](#), in §4.2.4

[attribute for HTMLAnchorElement](#), in §4.5.1

[attribute for HTMLAreaElement](#), in §4.7.16

[element-attr for a, links](#), in §4.8.2

hspace

[element-attr for embed](#), in §11.2

[element-attr for iframe](#), in §11.2

[element-attr for input](#), in §11.2

[element-attr for img](#), in §11.2

[element-attr for object](#), in §11.2

[attribute for HTMLAppletElement](#), in §11.3.1

[attribute for HTMLMarqueeElement](#), in §11.3.2

[attribute for HTMLImageElement](#), in §11.3.5

[attribute for HTMLObjectElement](#), in §11.3.5

[html](#), in §4.1.1

[HTMLAllCollection](#), in §2.7.2.1
[HTMLAnchorElement](#), in §4.5.1
[HTMLAppletElement](#), in §11.3.1
[HTMLAreaElement](#), in §4.7.16
[HTMLAudioElement](#), in §4.7.11
[HTMLBaseElement](#), in §4.2.3
[HTMLBodyElement](#), in §4.3.1
[HTMLBRElement](#), in §4.5.29
[HTMLButtonElement](#), in §4.10.6
[HTMLCanvasElement](#), in §4.12.4
[html-contents](#), in §3.2.4
[HTMLDataElement](#), in §4.5.15
[HTMLDataListElement](#), in §4.10.8
[HTMLDetailsElement](#), in §4.11.1
[HTMLDirectoryElement](#), in §11.3.5
[HTMLDivElement](#), in §4.4.14
[HTMLDLListElement](#), in §4.4.8
[HTMLDocument](#), in §6.3
[HTML document](#), in §2.1
[html element](#), in §2.1.2
[HTMLElement](#), in §3.2.2
[html elements](#), in §2.1.2
[HTMLEmbedElement](#), in §4.7.7
[HTMLFieldSetElement](#), in §4.10.16
[HTMLFontElement](#), in §11.3.5
[htmlFor](#)
 [attribute for HTMLLabelElement](#), in §4.10.4
 [attribute for HTMLOutputElement](#), in §4.10.13
 [attribute for HTMLScriptElement](#), in §11.3.5
[HTMLFormControlsCollection](#), in §2.7.2.2
[HTMLFormElement](#), in §4.10.3
[HTML fragment parsing algorithm](#), in §8.4
[HTML fragment serialization algorithm](#), in §8.3
[HTMLFrameElement](#), in §11.3.3
[HTMLFrameSetElement](#), in §11.3.3
[HTMLHeadElement](#), in §4.2.1
[HTMLHeadingElement](#), in §4.3.6
[HTMLHRElement](#), in §4.4.2
[HTMLHtmlElement](#), in §4.1.1
[HTMLHyperlinkElementUtils](#), in §4.8.3
[HTMLIFrameElement](#), in §4.7.6
[HTMLImageElement](#), in §4.7.5
[HTMLInputElement](#), in §4.10.5
[HTML integration point](#), in §8.2.5
[HTMLKeygenElement](#), in §4.10.12
[HTMLLabelElement](#), in §4.10.4
[HTMLLegendElement](#), in §4.10.17
[HTMLLIElement](#), in §4.4.7
[HTMLLinkElement](#), in §4.2.4
[html link types](#), in §4.8.6
[HTMLMapElement](#), in §4.7.15
[HTMLMarqueeElement](#), in §11.3.2
[HTMLMediaElement](#), in §4.7.14
[HTMLMenuElement](#), in §4.11.3
[HTMLMenuItemElement](#), in §4.11.4
[HTMLMetaElement](#), in §4.2.5
[HTMLMeterElement](#), in §4.10.15
[HTML MIME type](#), in §2.1.1
[HTMLModElement](#), in §4.6.3
[HTML namespace](#), in §2.8
[HTMLObjectElement](#), in §4.7.8
[HTMLLOListElement](#), in §4.4.5
[HTMLOptGroupElement](#), in §4.10.9
[HTMLOptionElement](#), in §4.10.10
[HTMLOptionsCollection](#), in §2.7.2.3
[HTMLOutputElement](#), in §4.10.13
[HTMLParagraphElement](#), in §4.4.1
[HTMLParamElement](#), in §4.7.9
[HTML parser](#), in §8.2
[HTMLPictureElement](#), in §4.7.3
[HTMLPreElement](#), in §4.4.3
[HTMLProgressElement](#), in §4.10.14
[HTMLQuoteElement](#), in §4.4.4
[HTMLScriptElement](#), in §4.12.1
[HTMLSelectElement](#), in §4.10.7
[HTMLSourceElement](#), in §4.7.12
[HTMLSpanElement](#), in §4.5.28
[HTMLStyleElement](#), in §4.2.6
[HTMLTableCaptionElement](#), in §4.9.2
[HTMLTableCellElement](#), in §4.9.11
[HTMLTableColElement](#), in §4.9.3
[HTMLTableDataCellElement](#), in §4.9.9
[HTMLTableElement](#), in §4.9.1
[HTMLTableHeaderCellElement](#), in §4.9.10
[HTMLTableRowElement](#), in §4.9.8
[HTMLTableSectionElement](#), in §4.9.5
[HTMLTemplateElement](#), in §4.12.3
[HTMLTextAreaElement](#), in §4.10.11
[HTMLTimeElement](#), in §4.5.16
[HTMLTitleElement](#), in §4.2.2
[HTMLTrackElement](#), in §4.7.13
[HTMLULListElement](#), in §4.4.6
[HTMLUnknownElement](#), in §3.2.2
[HTMLVideoElement](#), in §4.7.10
[http:](#), in §2.2.2
[http-equiv](#)
 [element-attr for meta](#), in §4.2.5
 [definition of](#), in §4.2.5.3
[httpEquiv](#), in §4.2.5
[HTTP GET method](#), in §2.6.1
[HTTP headers](#), in §2.6.1
[HTTP response codes](#), in §2.6.1
[https:](#), in §2.2.2
[HTTPS state](#)
 [dfn for document](#), in §3.1.1
 [dfn for settings](#), in §7.1.3.1
[hyperlink](#), in §4.8.1
[hyperlink annotations](#), in §4.8.1
[hyperlinks](#), in §4.8.1
[i](#)
 [attr-value for ol/type](#), in §4.4.5
 [\(element\)](#), in §4.5.22
[icon](#)
 [value for link/type](#), in §4.8.6.5
 [element-attr for menuitem](#), in §4.11.4
 [attribute for HTMLMenuItemElement](#), in §4.11.4
[id](#)
 [element-attr for global](#), in §3.2.5.1
 [attribute for AudioTrack](#), in §4.7.14.10.1
 [attribute for VideoTrack](#), in §4.7.14.10.1
 [attribute for TextTrack](#), in §4.7.14.11.5
 [attribute for TextTrackCue](#), in §4.7.14.11.5
[IDL attribute](#), in §2.1

- [IDL attributes](#), in §2.1
- [IDLE](#), in §11.3.4.8
- [IDL-exposed autofill value](#), in §4.10.19.8.2
- [if appropriate](#), in §5.7.4
- [iframe](#), in §4.7.6
- [iframe load event steps](#), in §4.7.6
- [iframe load in progress](#), in §4.7.6
- [iframe sandboxing flag set](#), in §6.5
- [iframe srcdoc document](#), in §4.7.6
- [ignored](#), in §2.1.3
- [ignore-destructive-writes counter](#), in §7.3.3
- [ignored ruby content](#), in §4.5.10
- [ignore higher-layer caching](#), in §4.7.5
- [ignore-opens-during-unload counter](#), in §7.3.1
- [image](#), in §4.10.5
- [ImageBitmap](#), in §7.7
- [ImageBitmapFactories](#), in §7.7
- [Image Button](#), in §4.10.5.1.19
- [image candidate string](#), in §4.7.5
- [image data](#), in §4.7.5
- [image format-based selection](#), in §4.7.1
- [image map](#), in §4.7.17
- [image maps](#), in §4.7.17
- [image request](#), in §4.7.5
- [images](#)
 - [attribute for Document](#), in §3.1.3
 - [attribute for HTMLMapElement](#), in §4.7.15
- [image sniffing](#), in §2.6.4
- [image sniffing rules](#), in §2.6.4
- [image source](#), in §4.7.5
- [Image\(width, height\)](#), in §4.7.5
- [img](#), in §4.7.5
- [immediately](#), in §2.1
- [immediate user selection](#), in §5.7.5
- [implementation notes](#), in §4.10.5.2
- [Implementation notes for session history](#), in §6.6.3
- [implement the sandboxing](#), in §6.5
- [implied](#), in §3.2.4.4
- [implied paragraph](#), in §3.2.4.4
- [implied paragraphs](#), in §3.2.4.4
- [implied strong reference](#), in §2.7.5
- [in a document](#), in §2.1.3
- [in a formal activation state](#), in §4.15.2
- [inappropriate for a control](#), in §4.10.19.8.1
- [inappropriate for the control](#), in §4.10.19.8.1
- [in-band metadata track dispatch type](#), in §4.7.14.11.1
- [inBandMetadataTrackDispatchType](#), in §4.7.14.11.5
- [in body](#), in §8.2.5.4.7
- [in caption](#), in §8.2.5.4.11
- [in cell](#), in §8.2.5.4.15
- [in column group](#), in §8.2.5.4.12
- [increment the marquee current loop index](#), in §11.3.2
- [Incumbent](#), in §7.1.3.5
- [incumbent global object](#), in §7.1.3.5.2
- [incumbent Realm](#), in §7.1.3.5.2
- [incumbent settings object](#), in §7.1.3.5.2
- [indeterminate](#), in §4.10.5
- [:indeterminate](#), in §4.15.2
- [index](#)
 - [dfn for option](#), in §4.10.10
 - [attribute for HTMLOptionElement](#), in §4.10.10
- [indexed for indexed property retrieval](#), in §4.10.3
- [indexed for named property retrieval](#), in §4.10.3
- [indicated a coordinate](#), in §4.10.5.1.19
- [indicated part of the document](#), in §6.7.9
- [in error reporting mode](#), in §7.1.3.8
- [inert](#), in §5.2
- [inertness](#), in §5.2
- [in foreign content](#), in §8.2.5.5
- [in frameset](#), in §8.2.5.4.20
- [in head](#), in §8.2.5.4.4
- [in head noscript](#), in §8.2.5.4.5
- [inherit-by-default](#), in §5.6.5
- [initial](#), in §8.2.5.4.1
- [Initializing a new Document object](#), in §6.7.1
- [initial playback position](#), in §4.7.14.6
- [initiated](#), in §5.7.5
- [initiate the drag-and-drop operation](#), in §5.7.5
- [in parallel](#), in §2.1
- [input](#)
 - [\(element\)](#), in §4.10.5
 - [event for input](#), in §4.10.5.5
 - [event for global](#), in §Unnumbered section
- [input byte stream](#), in §8.2.2
- [inputMode](#)
 - [attribute for HTMLInputElement](#), in §4.10.5
 - [attribute for HTMLTextAreaElement](#), in §4.10.11
- [inputmode](#)
 - [element-attr for textarea](#), in §4.10.11
 - [element-attr for input](#), in §4.10.19.7
- [input stream](#), in §8.2.2.5
- [:in-range](#), in §4.15.2
- [in row](#), in §8.2.5.4.14
- [ins](#), in §4.6.1
- [in scope](#), in §8.2.3.2
- [in select](#), in §8.2.5.4.16
- [in select in table](#), in §8.2.5.4.17
- [insert a character](#), in §8.2.5.1
- [insert a comment](#), in §8.2.5.1
- [insert a foreign element](#), in §8.2.5.1
- [insert an HTML element](#), in §8.2.5.1
- [insertCell\(index\)](#), in §4.9.8
- [insert characters](#), in §8.2.5.1
- [inserted into](#), in §2.1.3
- [inserted into a document](#), in §2.1.3
- [inserted into the document](#), in §2.1.3
- [insertion mode](#), in §8.2.3.1
- [insertion point](#), in §8.2.2.5
- [insertRow\(index\)](#)
 - [method for HTMLTableElement](#), in §4.9.1
 - [method for HTMLTableSectionElement](#), in §4.9.5
- [insert the character](#), in §8.2.5.1
- [insert the token's character](#), in §8.2.5.1
- [in table](#), in §8.2.5.4.9
- [in table body](#), in §8.2.5.4.13
- [in table scope](#), in §8.2.3.2

[in table text](#), in §8.2.5.4.10[in template](#), in §8.2.5.4.18[interactive](#), in §3.1.1["interactive"](#), in §3.1.1[interactive content](#), in §3.2.4.2.7[interactively validate the constraints](#), in §4.10.21.2[inter-element whitespace](#), in §3.2.4[internal algorithm for scanning and assigning header cells](#), in §4.9.12.2[internal pause steps](#), in §4.7.14.8[internal raw uncompiled handler](#), in §7.1.5.1[in the document](#), in §2.1.3[in the html parser](#), in §8.2.5.4.2[in the parser](#), in §8.2.5.4.2[intrinsic dimensions](#), in §2.2.2[intrinsic height](#)[dfn for css](#), in §2.2.2[dfn for video](#), in §4.7.10[intrinsic width](#)[dfn for css](#), in §2.2.2[dfn for video](#), in §4.7.10[invalid](#), in §Unnumbered section[:invalid](#), in §4.15.2[invalid value default](#), in §2.4.3[invoke](#), in §7.1.5.1[@@isConcatSpreadable](#), in §2.2.2[isContentEditable](#), in §5.6.1[isindex](#), in §11.2[isMap](#), in §4.7.5[ismap](#)[element-attr for img](#), in §4.7.5[element-attr for input](#), in §11.2[is not step aligned](#), in §4.10.5.4[ISO-2022-JP](#), in §8.2.2.3[ISO-8859-2](#), in §8.2.2.3[ISO-8859-8](#), in §8.2.2.3[IsPlatformObjectSameOrigin](#), in §6.2.3.2[IsSearchProviderInstalled\(\)](#), in §11.3.5[is step aligned](#), in §4.10.5.4[IsTransferable](#), in §2.9.5[it can also come from script](#), in §7.3[item\(\)](#), in §2.7.2.1[item\(index\)](#)[method for HTMLSelectElement](#), in §4.10.7[method for PluginArray](#), in §7.6.1.5[method for MimeTypeError](#), in §7.6.1.5[method for Plugin](#), in §7.6.1.5[item\(nameOrItem\)](#), in §2.7.2.1[items](#), in §5.7.3[item type string](#), in §5.7.2[javaEnabled\(\)](#), in §7.6.1.5[JavaScript MIME type](#), in §4.12.1.2[javascript: url](#), in §6.7.1[javascript: urls](#), in §6.7.1[joint session history](#), in §6.6.2[JSON MIME type](#), in §6.7.1[kana](#), in §4.10.19.7[Kana](#), in §4.10.19.7[kana-name](#), in §4.10.19.7[Kana Name](#), in §4.10.19.7[Katakana](#), in §4.10.19.7[katakana](#), in §4.10.19.7[kbd](#), in §4.5.20[keygen](#), in §4.10.12[keytype](#)[element-attr for keygen](#), in §4.10.12[attribute for HTMLKeygenElement](#), in §4.10.12[keywords](#), in §4.2.5.1[kind](#)[element-attr for track](#), in §4.7.13[attribute for HTMLTrackElement](#), in §4.7.13[attribute for AudioTrack](#), in §4.7.14.10.1[attribute for VideoTrack](#), in §4.7.14.10.1[definition of](#), in §4.7.14.11.1[attribute for TextTrack](#), in §4.7.14.11.5[attribute for DataTransferItem](#), in §5.7.3.2[kind of element](#), in §8.1.2[kind of track](#), in §4.7.14.11.1[kinds of elements](#), in §8.1.2[label](#)[element-attr for track](#), in §4.7.13[attribute for HTMLTrackElement](#), in §4.7.13[attribute for AudioTrack](#), in §4.7.14.10.1[attribute for VideoTrack](#), in §4.7.14.10.1[attribute for TextTrack](#), in §4.7.14.11.5[\(element\)](#), in §4.10.4[element-attr for optgroup](#), in §4.10.9[attribute for HTMLOptGroupElement](#), in §4.10.9[element-attr for option](#), in §4.10.10[definition of](#), in §4.10.10[attribute for HTMLOptionElement](#), in §4.10.10[element-attr for menu](#), in §4.11.3[attribute for HTMLMenuElement](#), in §4.11.3[element-attr for menuitem](#), in §4.11.4[attribute for HTMLMenuItemElement](#), in §4.11.4[Label](#), in §4.11.6.1[labelable](#), in §4.10.2[labelable element](#), in §4.10.2[labelable elements](#), in §4.10.2[labeled control](#), in §4.10.4[label of a track](#), in §4.7.14.11.1[labels](#), in §4.10.4[lack scripting support](#), in §2.2.1[lang](#)[element-attr for global](#), in §3.2.5.3[element-attr for xml](#), in §3.2.5.3[language](#)[definition of](#), in §3.2.5.3[attribute for AudioTrack](#), in §4.7.14.10.1[attribute for VideoTrack](#), in §4.7.14.10.1[attribute for TextTrack](#), in §4.7.14.11.5[attribute for NavigatorLanguage](#), in §7.6.1.2[element-attr for script](#), in §11.2[languagechange](#), in §Unnumbered section[language of a text track](#), in §4.7.14.11.1[languages](#), in §7.6.1.2[lastModified](#), in §3.1.2[last selected source](#), in §4.7.5

[latest entry](#), in §6.6.1[latin](#), in §4.10.19.7[Latin Name](#), in §4.10.19.7[latin-name](#), in §4.10.19.7[Latin Prose](#), in §4.10.19.7[latin-prose](#), in §4.10.19.7[Latin Text](#), in §4.10.19.7[Latin Verbatim](#), in §4.10.19.7[leading and trailing whitespace stripped](#), in §2.4.1[left](#)[attr-value for marquee/direction](#), in §10.5.11[state for marquee](#), in §11.3.2[leftmargin](#), in §11.2[legacy caller operation](#)[dfn for embed](#), in §4.7.7[dfn for object](#), in §4.7.8[legend](#), in §4.10.17[length](#)[attribute for HTMLAllCollection](#), in §2.7.2.1[attribute for HTMLOptionsCollection](#), in §2.7.2.3[attribute for AudioTrackList](#), in §4.7.14.10.1[attribute for VideoTrackList](#), in §4.7.14.10.1[attribute for TextTrackList](#), in §4.7.14.11.5[attribute for TextTrackCueList](#), in §4.7.14.11.5[attribute for TimeRanges](#), in §4.7.14.13[attribute for HTMLFormElement](#), in §4.10.3[attribute for HTMLSelectElement](#), in §4.10.7[attribute for DataTransferItemList](#), in §5.7.3.1[attribute for Window](#), in §6.3.2[attribute for History](#), in §6.6.2[attribute for PluginArray](#), in §7.6.1.5[attribute for MimeTypeArray](#), in §7.6.1.5[attribute for Plugin](#), in §7.6.1.5[li](#), in §4.4.7[license](#), in §4.8.6.6[limited-quirks mode](#), in §2.2.2[limited to numbers greater than zero](#), in §2.7.1[limited to only known values](#), in §2.7.1[limited to only non-negative numbers](#), in §2.7.1[limited to only non-negative numbers greater than zero](#), in §2.7.1[lineno](#), in §7.1.3.8.2[:link](#), in §4.15.2[link](#)[\(element\)](#), in §4.2.4[value for dropEffect](#), in §5.7.3[value for effectAllowed](#), in §5.7.3[value for drag](#), in §5.7.5[definition of](#), in §5.7.8[element-attr for body](#), in §11.2[attribute for HTMLBodyElement](#), in §11.3.5[linkColor](#), in §11.3.5[linkMove](#), in §5.7.3[links](#), in §3.1.3[links to external resources](#), in §4.8.1[link type](#), in §4.8.6[link types](#), in §4.8.6[list](#)[element-attr for input](#), in §4.10.5.3.9[attribute for HTMLInputElement](#), in §4.10.5.4[Listed element](#), in §4.10.2[Listed elements](#), in §4.10.2[listing](#), in §11.2[list of active formatting elements](#), in §8.2.3.3[list of active timers](#), in §7.4[list of animation frame callbacks](#), in §7.8[list of available images](#), in §4.7.5[list of cues](#), in §4.7.14.11.1[list of cues of a text track](#), in §4.7.14.11.1[list of dragged nodes](#), in §5.7.5[list of newly introduced cues](#), in §4.7.14.8[list of options](#), in §4.10.7[list of pending master entries](#), in §11.3.4[list of pending text tracks](#), in §4.7.14.11.1[list of scripts that will execute in order as soon as possible](#), in §4.12.1.1[list of scripts that will execute when the document has finished parsing](#), in §4.12.1.1[list of text tracks](#), in §4.7.14.11.1[list of the descendant browsing contexts](#), in §6.1.1[live](#), in §2.1.4[load\(\)](#)[\(function\)](#), in §3.1.4[method for HTMLMediaElement](#), in §4.7.14.5[load](#)[event for track](#), in §4.7.14.15[event for global](#), in §Unnumbered section[loaded](#), in §4.7.14.11.1[LOADED](#), in §4.7.13[loadeddata](#), in §4.7.14.15[loadedmetadata](#), in §4.7.14.15[loadend](#), in §Unnumbered section[LOADING](#), in §4.7.13[loading](#)[enum-value for DocumentReadyState](#), in §3.1.1[state for track](#), in §4.7.14.11.1["loading"](#), in §3.1.1[loadstart](#)[event for media](#), in §4.7.14.15[event for global](#), in §Unnumbered section[load\(url\)](#), in §3.1.4[Local Date and Time](#), in §4.10.5.1.11[Location](#), in §6.6.4[location](#)[attribute for Document](#), in §6.6.4[attribute for Window](#), in §6.6.4[locationbar](#), in §6.3.6[Location-object navigate](#), in §6.6.4[Location-object-setter navigate](#), in §6.6.4[locked for focus](#), in §5.4.6[locked for reset](#), in §4.10.3

loop

[element-attr for media](#), in §4.7.14.6

[attribute for HTMLMediaElement](#), in §4.7.14.6

[element-attr for marquee](#), in §11.3.2

[attribute for HTMLMarqueeElement](#), in §11.3.2

loses focus, in §5.4.4**low**

[element-attr for meter](#), in §4.10.15

[attribute for HTMLMeterElement](#), in §4.10.15

low boundary, in §4.10.15**lower-alpha**, in §4.4.5**lowercase ASCII hex digits**, in §2.4.1**lowercase ASCII letters**, in §2.4.1**lower-roman**, in §4.4.5**lowsrc**

[element-attr for img](#), in §11.2

[attribute for HTMLImageElement](#), in §11.3.5

ltr

[attr-value for global/dir](#), in §3.2.5.6

[state for dir](#), in §3.2.5.6

LTR-specific, in §10.2**machine-readable equivalent of the element's contents**, in §4.5.16**Mail as body**, in §4.10.22.3**mailto:**, in §2.2.2**Mail with headers**, in §4.10.22.3**main**

[\(element\)](#), in §4.4.13

[attr-value for commonTrack/kind](#), in §4.7.14.10.1

main-desc, in §4.7.14.10.1**manifest**

[element-attr for html](#), in §11.2

[definition of](#), in §11.3.4

manual

[value for scrollRestorationMode](#), in §6.6.1

[enum-value for ScrollRestoration](#), in §6.6.2

"manual", in §6.6.2**map**, in §4.7.15**maps to the dimension properties**, in §10.2**maps to the dimension property**, in §10.2**maps to the dimension property (ignoring zero)**, in §10.2**maps to the pixel length property**, in §10.2**map to the dimension property (ignoring zero)**, in §10.2**marginHeight**

[attribute for HTMLFrameElement](#), in §11.3.3

[attribute for HTMLIFrameElement](#), in §11.3.5

marginheight

[element-attr for body](#), in §11.2

[element-attr for iframe](#), in §11.2

margin-top, in §11.2**marginwidth**

[element-attr for body](#), in §11.2

[element-attr for iframe](#), in §11.2

marginWidth

[attribute for HTMLFrameElement](#), in §11.3.3

[attribute for HTMLIFrameElement](#), in §11.3.5

mark, in §4.5.25**markers**, in §8.2.3.3**marquee**, in §11.3.2**marquee current loop index**, in §11.3.2**marquee loop count**, in §11.3.2**marquee scroll distance**, in §11.3.2**marquee scroll interval**, in §11.3.2**master**, in §11.3.4**master entries**, in §11.3.4**matches a drag data store**, in §5.7.8**matches a fallback namespace**, in §11.3.4**matches the environment**, in §2.4.10**matches the fallback namespace**, in §11.3.4**match service worker registration**, in §2.2.2**match the environment**, in §2.4.10**MathML namespace**, in §2.8**MathML text integration point**, in §8.2.5**matured**, in §6.7.1**max**

[attribute for HTMLInputElement](#), in §4.10.5

[element-attr for input](#), in §4.10.5.3.7

[element-attr for progress](#), in §4.10.14

[attribute for HTMLProgressElement](#), in §4.10.14

[element-attr for meter](#), in §4.10.15

[attribute for HTMLMeterElement](#), in §4.10.15

maximum, in §4.10.5.3.7**maximum allowed value length**, in §4.10.19.3**maximum value**

[dfn for progress](#), in §4.10.14

[dfn for meter](#), in §4.10.15

maxLength

[attribute for HTMLInputElement](#), in §4.10.5

[attribute for HTMLTextAreaElement](#), in §4.10.11

maxlength

[element-attr for input](#), in §4.10.5.3.1

[element-attr for textarea](#), in §4.10.11

"maybe", in §4.7.14**maybe**

[enum-value for CanPlayTypeResult](#), in §4.7.14

[definition of](#), in §4.7.14.3

media

[element-attr for link](#), in §4.2.4

[attribute for HTMLLinkElement](#), in §4.2.4

[element-attr for style](#), in §4.2.6

[attribute for HTMLStyleElement](#), in §4.2.6

[element-attr for source](#), in §4.7.4

media data, in §4.7.14**media data processing steps list**, in §4.7.14.5**media element**, in §4.7.14**media element attributes**, in §4.7.14**media element event task source**, in §4.7.14**media element load algorithm**, in §4.7.14.5**media elements**, in §4.7.14**MEDIA_ERR_ABORTED**, in §4.7.14.1**MEDIA_ERR_DECODE**, in §4.7.14.1

[MEDIA_ERR_NETWORK](#), in

§4.7.14.1

[MediaError](#), in §4.7.14.1

[MEDIA_ERR_SRC_NOT_SUPPORTED](#), in

§4.7.14.1

[media provider object](#), in §4.7.14.2

[media resource](#), in §4.7.14

[media-resource-specific text track](#), in §4.7.14.11.2

[media timeline](#), in §4.7.14.6

[media type](#), in §2.1.1

menu

[attr-value for button/type](#), in §4.10.6

[element-attr for button](#), in §4.10.6

[attribute for HTMLButtonElement](#), in §4.10.6

[\(element\)](#), in §4.11.3

[Menu](#), in §4.10.6

[menubar](#), in §6.3.6

[menu button](#), in §4.10.6

[menu command](#), in §4.11.6.5

[menu construct](#), in §4.11.3

[menuitem](#), in §4.11.4

[menu item constructs](#), in §4.11.3

[menu item generator](#), in §4.11.3

message

[attribute for ErrorEvent](#), in §7.1.3.8.2

[event for global](#), in §Unnumbered section

[meta](#), in §4.2.5

Metadata

[state for track](#), in §4.7.13

[state for media](#), in §4.7.14.5

["metadata"](#), in §4.7.14.11.5

metadata

[attr-value for track/kind](#), in §4.7.13

[value for](#)

[HTMLMediaElement/preload](#), in §4.7.14.5

[definition of](#), in §4.7.14.11.1

[enum-value for TextTrackKind](#), in §4.7.14.11.5

[metadata content](#), in §3.2.4.2.1

[metadata names](#), in §4.2.5.2

[meter](#), in §4.10.15

method

[element-attr for form](#), in §4.10.19.6

[dfn for forms](#), in §4.10.19.6

[attribute for HTMLFormElement](#), in §4.10.19.6

methods

[element-attr for a](#), in §11.2

[element-attr for link](#), in §11.2

[microtask](#), in §7.1.4.2

[microtask checkpoints](#), in §7.1.4.2

[microtask queue](#), in §7.1.4.2

[microtask task source](#), in §7.1.4.2

[mime type](#), in §2.1.1

[MimeType](#), in §7.6.1.5

[MimeTypeArray](#), in §7.6.1.5

[mimeTypes](#), in §7.6.1.5

[mime types](#), in §2.1.1

min

[attribute for HTMLInputElement](#), in §4.10.5

[element-attr for input](#), in §4.10.5.3.7

[element-attr for meter](#), in §4.10.15

[attribute for HTMLMeterElement](#), in §4.10.15

[minimum](#), in §4.10.5.3.7

[minimum allowed value length](#), in §4.10.19.4

[minimum value](#), in §4.10.15

minLength

[attribute for HTMLInputElement](#), in §4.10.5

[attribute for HTMLTextAreaElement](#), in §4.10.11

minlength

[element-attr for input](#), in §4.10.5.3.1

[element-attr for textarea](#), in §4.10.11

[missing value default](#), in §2.4.3

[mode](#), in §4.7.14.11.5

[Month](#), in §4.10.5.1.8

month

[definition of](#), in §2.4.5.1

[attr-value for input/type](#), in §4.10.5

[most appropriate application cache](#), in §11.3.4

move

[value for dropEffect](#), in §5.7.3

[value for effectAllowed](#), in §5.7.3

[value for drag](#), in §5.7.5

[definition of](#), in §5.7.8

[multicol](#), in §11.2

[multipart/form-data](#), in §4.10.19.6

[multipart/form-data boundary string](#), in §4.10.22.7

[multipart/form-data encoding algorithm](#), in §4.10.22.7

multiple

[attribute for HTMLInputElement](#), in §4.10.5

[element-attr for input](#), in §4.10.5.3.5

[element-attr for select](#), in §4.10.7

[attribute for HTMLSelectElement](#), in §4.10.7

[mutable](#), in §4.10.18.2

[Mutate action URL](#), in §4.10.22.3

[mutation observers](#), in §2.2.2

muted

[definition of](#), in §4.7.14.12

[attribute for HTMLMediaElement](#), in §4.7.14.12

[element-attr for media](#), in §4.7.14.12

[muted errors](#), in §7.1.3.1

[mute iframe load](#), in §4.7.6

name	namedItem(name)	newer , in §11.3.4
event for global , in §2.2.2	method for HTMLAllCollection , in §2.7.2.1	newest , in §11.3.4
element-attr for meta , in §4.2.5	method for HTMLFormControlsCollection , in §2.7.2.2	newURL , in §6.7.10.3
attribute for HTMLMetaElement , in §4.2.5	method for HTMLSelectElement , in §4.10.7	next , in §4.8.6.12.1
element-attr for iframe , in §4.7.6	method for PluginArray , in §7.6.1.5	nextid , in §11.2
attribute for HTMLIFrameElement , in §4.7.6	method for MimeTypeArray , in §7.6.1.5	next input character , in §8.2.2.5
element-attr for object , in §4.7.8	method for Plugin , in §7.6.1.5	next token , in §8.2.5
element-attr for param , in §4.7.9	Named objects , in §6.3.3	nobr , in §11.2
attribute for HTMLParamElement , in §4.7.9	names , in §8.1.2.3	No CORS , in §2.6.6
element-attr for map , in §4.7.15	naturalHeight , in §4.7.5	node A is removed , in §2.1.3
attribute for HTMLMapElement , in §4.7.15	naturalWidth , in §4.7.5	noembed , in §11.2
element-attr for form , in §4.10.3	nav , in §4.3.4	nofollow , in §4.8.6.7
attribute for HTMLFormElement , in §4.10.3	navigate , in §6.7.1	noframes , in §11.2
element-attr for formelements, label, input, button, select, textarea, keygen, output, fieldset , in §4.10.19.1	navigated , in §6.7.1	nohref , in §11.2
attribute for HTMLInputElement, HTMLButtonElement, HTMLSelectElement, HTMLTextAreaElement, HTMLKeygenElement, HTMLOutputElement, HTMLFieldSetElement , in §4.10.19.1	navigating , in §6.7.1	noHref , in §11.3.5
attribute for Window , in §6.3.1	navigating a browsing context , in §6.7.1	"non-blocking" , in §4.12.1.1
attribute for Plugin , in §7.6.1.5	navigation , in §6.7.1	nonce
definition of , in §8.1.2.3	navigation algorithm , in §6.7.1	element-attr for style , in §4.2.6
element-attr for a , in §11.2	Navigator , in §7.6.1	attribute for HTMLStyleElement , in §4.2.6
element-attr for embed , in §11.2	navigator , in §7.6.1	element-attr for script , in §4.12.1
element-attr for img , in §11.2	NavigatorContentUtils , in §7.6.1.3	attribute for HTMLScriptElement , in §4.12.1
element-attr for option , in §11.2	NavigatorCookies , in §7.6.1.4	none
attribute for HTMLAppletElement , in §11.3.1	NavigatorID , in §7.6.1.1	value for HTMLMediaElement/preload , in §4.7.14.5
element-attr for frame , in §11.3.3	NavigatorLanguage , in §7.6.1.2	context for canvas , in §4.12.4
attribute for HTMLFrameElement , in §11.3.3	navigator.onLine , in §6.7.13	value for dropEffect , in §5.7.3
attribute for HTMLAnchorElement , in §11.3.5	NavigatorOnLine , in §6.7.13	value for effectAllowed , in §5.7.3
attribute for HTMLEmbedElement , in §11.3.5	NavigatorPlugins , in §7.6.1.5	value for drag , in §5.7.5
attribute for HTMLImageElement , in §11.3.5	nearest activatable element , in §5.3	None , in §4.7.14.5
named color , in §2.2.2	nearest ancestor autofocus scoping root element , in §4.10.19.6.1	NONE , in §4.7.13
named colors , in §2.2.2	nested browsing context , in §6.1.1	no-quirks mode , in §2.2.2
Named elements , in §3.1.3	nested browsing contexts , in §6.1.1	noreferrer , in §4.8.6.8
named for the all collection , in §2.7.2.1	nested through , in §6.1.1	noResize , in §11.3.3
	NETWORK EMPTY , in §4.7.14.4	normal elements , in §8.1.2
	NETWORK IDLE , in §4.7.14.4	normalized TimeRanges object , in §4.7.14.13
	networking task source , in §7.1.4.3	normalize the source densities , in §4.7.5
	NETWORK LOADING , in §4.7.14.4	noscript , in §4.12.2
	NETWORK NO SOURCE , in §4.7.14.4	noShade , in §11.3.5
	networkState , in §4.7.14.4	noshade , in §11.2
	new , in §2.7.4	not handled
		dfn for script , in §7.1.3.8
		dfn for promise , in §7.1.3.9
		nothing , in §3.2.4.1

[notify about rejected promises](#), in §7.1.3.9

[not loaded](#), in §4.7.14.11.1

[no-translate](#), in §3.2.5.4

[not yet been loaded](#), in §4.7.14.11.1

[novalidate](#), in §4.10.19.6

[noValidate](#), in §4.10.19.6

[no-validate state](#), in §4.10.19.6

[nowrap](#), in §11.2

[noWrap](#), in §11.3.5

[number](#), in §4.10.5

[Number](#), in §4.10.5.1.12

[number of bytes downloaded](#), in §2.6.2

[number of child browsing contexts](#), in §6.3.2

[number of days in month month of year year](#), in §2.4.5

[numeric](#), in §4.10.19.7

[Numeric](#), in §4.10.19.7

[object](#)

(element), in §4.7.8

[attribute for HTMLAppletElement](#), in §11.3.1

[object](#), in §11.3.1

[object properties](#), in §2.1

[object property](#), in §2.1

[OBSOLETE](#), in §11.3.4.8

[obsolete](#), in §11.3.4

[obsolete permitted DOCTYPE](#), in §8.1.1

[obsolete permitted DOCTYPE string](#), in §8.1.1

[obtain](#), in §4.2.4

[obtain a physical form](#), in §10.8

[obtain the resource](#), in §4.2.4

[off](#)

[attr-value for form/autocomplete](#), in §4.10.3

[state for form/autocomplete](#), in §4.10.3

[attr-value for forms/autocomplete](#), in §4.10.19.8.1

[official playback position](#), in §4.7.14.6

[offline](#), in §Unnumbered section

[ol](#), in §4.4.5

[oldURL](#), in §6.7.10.3

[omitted](#), in §8.1.2.4

[on](#)

[attr-value for form/autocomplete](#), in §4.10.3

[state for form/autocomplete](#), in §4.10.3

[attr-value for forms/autocomplete](#), in §4.10.19.8.1

[onabort](#), in §7.1.5.2

[onaddtrack](#)

[attribute for AudioTrackList](#), in §4.7.14.10.1

[attribute for VideoTrackList](#), in §4.7.14.10.1

[attribute for MediaTrackList](#), in §4.7.14.10.1

[attribute for TextTrackList](#), in §4.7.14.11.8

[onafterprint](#), in §7.1.5.2

[onbeforeprint](#), in §7.1.5.2

[onbeforeunload](#)

[attribute for OnBeforeUnloadEventHandler](#), in §7.1.5.2

[attribute for WindowEventHandlers](#), in §7.1.5.2.1

[OnBeforeUnloadEventHandler](#), in §7.1.5.1

[OnBeforeUnloadEventHandlerNonNull](#), in §7.1.5.1

[onblur](#), in §7.1.5.2

[onbounce](#), in §11.3.2

[oncached](#), in §11.3.4.8

[oncancel](#), in §7.1.5.2

[oncanplay](#), in §7.1.5.2

[oncanplaythrough](#), in §7.1.5.2

[onchange](#)

[attribute for AudioTrackList](#), in §4.7.14.10.1

[attribute for VideoTrackList](#), in §4.7.14.10.1

[attribute for MediaTrackList](#), in §4.7.14.10.1

[attribute for TextTrackList](#), in §4.7.14.11.8

[attribute for GlobalEventHandlers](#), in §7.1.5.2

[onchecking](#), in §11.3.4.8

[onclick](#), in §7.1.5.2

[onclose](#), in §7.1.5.2

[oncontextmenu](#), in §7.1.5.2

[oncopy](#), in §7.1.5.2

[oncuechange](#)

[attribute for TextTrack](#), in §4.7.14.11.8

[attribute for GlobalEventHandlers](#), in §7.1.5.2

[oncut](#), in §7.1.5.2

[ondblclick](#), in §7.1.5.2

[on downloading](#), in §11.3.4.8

[ondrag](#), in §7.1.5.2

[ondragend](#), in §7.1.5.2

[ondragenter](#), in §7.1.5.2

[ondragexit](#), in §7.1.5.2

[ondragleave](#), in §7.1.5.2

[ondragover](#), in §7.1.5.2

[ondragstart](#), in §7.1.5.2

[ondrop](#), in §7.1.5.2

[ondurationchange](#), in §7.1.5.2

[onemptied](#), in §7.1.5.2

[onended](#), in §7.1.5.2

[onenter](#), in §4.7.14.11.8

[one permitted sandboxed navigator](#), in §6.5

[onerror](#)

[attribute for GlobalEventHandlers](#), in §7.1.5.2

[attribute for ApplicationCache](#), in §11.3.4.8

[OnErrorEventHandler](#), in §7.1.5.1

[OnErrorEventHandlerNonNull](#), in §7.1.5.1

[onexit](#), in §4.7.14.11.8

[onfinish](#), in §11.3.2

[onfocus](#), in §7.1.5.2

[onhashchange](#), in §7.1.5.2

[oninput](#), in §7.1.5.2

[oninvalid](#), in §7.1.5.2

[onkeydown](#), in §7.1.5.2

[onkeypress](#), in §7.1.5.2

[onkeyup](#), in §7.1.5.2

[onlanguagechange](#), in §7.1.5.2

[online](#), in §Unnumbered section

[onLine](#), in §6.7.13

[online safelist](#), in §11.3.4.1

[online safelist namespaces](#), in §11.3.4

[online safelist wildcard flag](#), in §11.3.4

[onload](#), in §7.1.5.2

[onloadeddata](#), in §7.1.5.2[onloadedmetadata](#), in §7.1.5.2[onloadstart](#), in §7.1.5.2[only if border is not equivalent to zero](#), in §10.3.9[onmessage](#), in §7.1.5.2[onmousedown](#), in §7.1.5.2[onmouseenter](#), in §7.1.5.2[onmouseleave](#), in §7.1.5.2[onmousemove](#), in §7.1.5.2[onmouseout](#), in §7.1.5.2[onmouseover](#), in §7.1.5.2[onmouseup](#), in §7.1.5.2[onnoupdate](#), in §11.3.4.8[onobsolete](#), in §11.3.4.8[onoffline](#), in §7.1.5.2[ononline](#), in §7.1.5.2[onpagehide](#), in §7.1.5.2[onpageshow](#), in §7.1.5.2[onpaste](#), in §7.1.5.2[onpause](#), in §7.1.5.2[onplay](#), in §7.1.5.2[onplaying](#), in §7.1.5.2[onpopstate](#), in §7.1.5.2[onprogress](#)[attribute for GlobalEventHandlers](#), in §7.1.5.2[attribute for ApplicationCache](#), in §11.3.4.8[onratechange](#), in §7.1.5.2[onreadystatechange](#)[attribute for Document](#), in §3.1.1[definition of](#), in §7.1.5.2[onrejectionhandled](#), in §7.1.5.2[onremovetrack](#)[attribute for AudioTrackList](#), in §4.7.14.10.1[attribute for VideoTrackList](#), in §4.7.14.10.1[attribute for MediaTrackList](#), in §4.7.14.10.1[attribute for TextTrackList](#), in §4.7.14.11.8[onreset](#), in §7.1.5.2[onresize](#), in §7.1.5.2[onscroll](#), in §7.1.5.2[onseeked](#), in §7.1.5.2[onseeking](#), in §7.1.5.2[onselect](#), in §7.1.5.2[onshow](#), in §7.1.5.2[onstalled](#), in §7.1.5.2[onstart](#), in §11.3.2[onstorage](#), in §7.1.5.2[onsubmit](#), in §7.1.5.2[onsuspend](#), in §7.1.5.2[ontimeupdate](#), in §7.1.5.2[ontoggle](#), in §7.1.5.2[onunhandledrejection](#), in §7.1.5.2[onunload](#), in §7.1.5.2[onupdateready](#), in §11.3.4.8[onvolumechange](#), in §7.1.5.2[onwaiting](#), in §7.1.5.2[onwheel](#), in §7.1.5.2[opaque origin](#), in §6.4[open](#)[element-attr for details](#), in §4.11.1[attribute for HTMLDetailsElement](#), in §4.11.1[event for global](#), in §Unnumbered section[open\(\)](#)[method for Window](#), in §6.3.1[method for Document](#), in §7.3.1[opener](#)[definition of](#), in §6.1.2.1[attribute for Window](#), in §6.3[opener browsing context](#), in §6.1.2[open\(type\)](#), in §7.3.1[open\(type, replace\)](#), in §7.3.1[open\(url, name, features\)](#), in §7.3.1[open\(url, name, features, replace\)](#), in §7.3.1[optgroup](#), in §4.10.9[optimum](#)[element-attr for meter](#), in §4.10.15[attribute for HTMLMeterElement](#), in §4.10.15[optimum point](#), in §4.10.15[optimum value](#), in §4.10.15[option](#), in §4.10.10[:optional](#), in §4.15.2[optionally truncate a simple dialog string](#), in §7.5.1[optionally truncated](#), in §7.5.1[optionally truncating](#), in §7.5.1[optional start and end tags](#), in §8.1.2.4[options](#)[attribute for HTMLSelectElement](#), in §4.10.7[attribute for HTMLDataListElement](#), in §4.10.8[Option\(text, value, defaultSelected, selected\)](#), in §4.10.10[ordered set of unique space-separated tokens](#), in §2.4.7[ordinal value](#), in §4.4.7[Ordinary](#), in §8.2.3.2[or equivalent](#), in §2.6.1[origin](#)[attribute for HTMLHyperlinkElementUtils](#), in §4.8.3[dfn for concept](#), in §6.4[attribute for Location](#), in §6.6.4[original insertion mode](#), in §8.2.3.1[origin-clean](#), in §4.12.4[origin domain](#), in §6.4[origin host](#), in §6.4[origin port](#), in §6.4[origins](#), in §6.4[origin scheme](#), in §6.4[Other link types](#), in §4.8.6.13[outline](#), in §4.3.10.1[outline depth](#), in §4.3.10.1[:out-of-range](#), in §4.15.2[output](#), in §4.10.13[outstanding rejected promises weak set](#), in §7.1.3.1[overridden reload](#), in §3.1[override URL](#), in §6.7.1[owner](#), in §5.4.2[p](#), in §4.4.1[pagehide](#), in §Unnumbered section[pageshow](#), in §Unnumbered section[page showing](#), in §6.7.11[PageTransitionEvent](#), in §6.7.10.4[PageTransitionEvent\(type\)](#), in §6.7.10.4[PageTransitionEvent\(type, eventInitDict\)](#), in §6.7.10.4[palpable content](#), in §3.2.4.2.8[paragraph](#), in §3.2.4.4

- [paragraphing](#), in §3.2.4.4
- [paragraphs](#), in §3.2.4.4
- [param](#), in §4.7.9
- [parameter](#), in §4.7.9
- [parent](#), in §6.1.1.1
- [parent browsing context](#), in §6.1.1
- [parse](#), in §2.5.2
- [parse a date component](#), in §2.4.5.2
- [parse a date or time string](#), in §2.4.5.10
- [parse a date string](#), in §2.4.5.2
- [parse a duration string](#), in §2.4.5.9
- [parse a floating date and time string](#), in §2.4.5.5
- [parse a global date and time string](#), in §2.4.5.7
- [parse a manifest](#), in §11.3.4.1
- [parse a month component](#), in §2.4.5.1
- [parse a month string](#), in §2.4.5.1
- [parse a sandboxing directive](#), in §6.5
- [parse a sizes attribute](#), in §4.7.5
- [parse a srcset attribute](#), in §4.7.5
- [parse a time component](#), in §2.4.5.4
- [parse a time string](#), in §2.4.5.4
- [parse a time-zone offset component](#), in §2.4.5.6
- [parse a time-zone offset string](#), in §2.4.5.6
- [parse a URL](#), in §2.5.2
- [parse a week string](#), in §2.4.5.8
- [parse a yearless date component](#), in §2.4.5.3
- [parse a yearless date string](#), in §2.4.5.3
- [parse child's sizes attribute](#), in §4.7.5
- [parse child's srcset attribute](#), in §4.7.5
- [parsed as a CSS <color> value](#), in §2.2.2
- [parse error](#), in §8.2
- [parse errors](#), in §8.2
- [parse it as an integer](#), in §2.4.4.1
- ["parser-inserted"](#), in §4.12.1.1
- [parser pause flag](#), in §8.2.1
- [parse that attribute's value](#), in §2.4.4.2
- [parse the sandboxing directive](#), in §6.5
- [parse token as an integer](#), in §2.4.4.1
- [parsing](#), in §2.5.2
- [parsing a date](#), in §2.4.5.2
- [parsing a date and time](#), in §2.4.5.7
- [parsing a date string](#), in §2.4.5.2
- [parsing a duration string](#), in §2.4.5.9
- [parsing a floating date and time](#), in §2.4.5.5
- [parsing a floating date and time string](#), in §2.4.5.5
- [parsing a month](#), in §2.4.5.1
- [parsing a month string](#), in §2.4.5.1
- [parsing a time](#), in §2.4.5.4
- [parsing a time string](#), in §2.4.5.4
- [parsing a time-zone offset string](#), in §2.4.5.6
- [parsing a week](#), in §2.4.5.8
- [parsing a week string](#), in §2.4.5.8
- [parsing a yearless date string](#), in §2.4.5.3
- [parsing of relative urls](#), in §2.5.2
- [parsing relative urls](#), in §2.5.2
- [partially available](#), in §4.7.5
- [Partially available](#), in §4.7.5
- [passing its URL or data to an external software package](#), in §6.7.1
- [Password](#), in §4.10.5.1.6
- [password](#)
 - [attribute for HTMLHyperlinkElementUtils](#), in §4.8.3
 - [attr-value for input/type](#), in §4.10.5
- [paste](#), in §Unnumbered section
- [past names map](#), in §4.10.3
- [Path components](#), in §4.10.5.1.17
- [pathname](#)
 - [attribute for HTMLHyperlinkElementUtils](#), in §4.8.3
 - [attribute for Location](#), in §6.6.4
- [pattern](#)
 - [attribute for HTMLInputElement](#), in §4.10.5
 - [element-attr for input](#), in §4.10.5.3.6
- [patternMismatch](#), in §4.10.21.3
- [pause\(\)](#), in §4.7.14.8
- [Pause](#), in §7.1.4.2
- [pause](#)
 - [event for media](#), in §4.7.14.15
 - [state for useragent](#), in §7.1.4.2
- [paused](#), in §4.7.14.8
- [paused for in-band content](#), in §4.7.14.8
- [paused for user interaction](#), in §4.7.14.8
- [pause-on-exit](#), in §4.7.14.11.1
- [pauseOnExit](#), in §4.7.14.11.5
- [pause-on-exit flag](#), in §4.7.14.11.1
- [pending application cache download process tasks](#), in §11.3.4.2
- [pending parsing-blocking script](#), in §4.12.1.1
- [pending request](#), in §4.7.5
- [pending table character tokens](#), in §8.2.5.4.9
- [pending text track change notification flag](#), in §4.7.14.11.1
- [perform a microtask checkpoint](#), in §7.1.4.2
- [perform automatic text track selection](#), in §4.7.14.11.3
- [performing a microtask checkpoint flag](#), in §7.1.4.1
- [performs a microtask checkpoint](#), in §7.1.4.2
- [persisted](#), in §6.7.10.4
- [personalbar](#), in §6.3.6
- [phrasing content](#), in §3.2.4.2.5
- [pick an encoding for a form](#), in §4.10.22.5
- [picked](#), in §4.10.7
- [picking an encoding for the form](#), in §4.10.22.5
- [picture](#), in §4.7.3
- [picture source](#), in §4.7.4
- [picture source type](#), in §4.7.4
- [placeholder](#)
 - [attribute for HTMLInputElement](#), in §4.10.5
 - [element-attr for input](#), in §4.10.5.3.10
 - [element-attr for textarea](#), in §4.10.11
 - [attribute for HTMLTextAreaElement](#), in §4.10.11
- [placeholder label option](#), in §4.10.7
- [plaintext](#), in §11.2

- [plain text file](#), in §6.7.4
- [planned navigation](#), in §4.10.22.3
- [plan to navigate](#), in §4.10.22.3
- [platform](#), in §7.6.1.1
- [plausible languages](#), in §7.6.1.2
- [play](#), in §4.7.14.15
- [play\(\)](#), in §4.7.14.8
- [playback ended](#), in §4.7.14.8
- [playback has ended](#), in §4.7.14.8
- [playbackRate](#), in §4.7.14.8
- [playback volume](#), in §4.7.14.12
- [played](#), in §4.7.14.8
- [playing](#), in §4.7.14.15
- [Plugin](#), in §7.6.1.5
- [plugin](#), in §2.1.5
- [PluginArray](#), in §7.6.1.5
- [plugin document](#), in §6.7.7
- [plugins](#)
 - [definition of](#), in §2.1.5
 - [attribute for Document](#), in §3.1.3
 - [attribute for NavigatorPlugins](#), in §7.6.1.5
- [poly](#), in §4.7.16
- [polygon](#), in §4.7.16
- [Polygon state](#), in §4.7.16
- [polygon state](#), in §4.7.16
- [popstate](#), in §Unnumbered section
- [PopStateEvent](#), in §6.7.10.2
- [PopStateEvent\(type\)](#), in §6.7.10.2
- [PopStateEvent\(type, eventInitDict\)](#), in §6.7.10.2
- [populate the list of pending text tracks](#), in §4.7.14.11.1
- [popup menu](#), in §4.11.3
- [popup sandboxing flag set](#), in §6.5
- [port](#)
 - [attribute for HTMLHyperlinkElementUtils](#), in §4.8.3
 - [attribute for Location](#), in §6.6.4
- [position](#), in §4.10.14
- [possibly appropriate alternatives](#), in §5.7.4
- [post](#), in §4.10.19.6
- [POST](#), in §4.10.19.6
- [poster](#)
 - [element-attr for video](#), in §4.7.10
 - [definition of](#), in §4.7.10
- [poster frame](#), in §4.7.10
- [Post to data:](#), in §4.10.22.3
- [potentially active](#), in §4.7.7
- [potentially playing](#), in §4.7.14.8
- [practical concerns](#), in §2.2.1
- [pragma-set default language](#), in §4.2.5.3
- [pre](#), in §4.4.3
- [pre-click activation steps](#), in §5.3
- [prefer-online](#), in §11.3.4
- [prefix match](#), in §2.3
- [prefix match patterns](#), in §11.3.4
- [preload](#)
 - [element-attr for media](#), in §4.7.14.5
 - [attribute for HTMLMediaElement](#), in §4.7.14.5
- [prepare an event](#), in §4.7.14.8
- [prepare a script](#), in §4.12.1.1
- [prepared](#), in §4.12.1.1
- [prepare to run script](#), in §7.1.3.4
- [preparing](#), in §4.12.1.1
- [prescan a byte stream to determine its encoding](#), in §8.2.2.2
- [prescan the byte stream to determine its encoding](#), in §8.2.2.2
- [presentational hints](#), in §10.2
- [preserve](#)
 - [enum-value for SelectionMode](#), in §4.10.20
 - [enum for SelectionMode](#), in §4.10.20
- ["preserve"](#), in §4.10.20
- [prev](#), in §4.8.6.12.2
- [prevents content from creating new auxiliary browsing contexts](#), in §6.1.5
- [previous target element](#), in §5.7.5
- [primary control group](#), in §5.4.5
- [print\(\)](#), in §7.5.2
- [printing steps](#), in §7.5.2
- [print when loaded](#), in §7.5.2
- [probably](#)
 - [enum-value for CanPlayTypeResult](#), in §4.7.14
 - [definition of](#), in §4.7.14.3
- ["probably"](#), in §4.7.14
- [probablySupportsContext\(contextId, arguments...\)](#), in §4.12.4
- [probablySupportsContext\(contextId, arguments\)](#), in §4.12.4
- [proceed with that mechanism instead](#), in §6.7.1
- [process an rtc element](#), in §4.5.13
- [processing model for navigating across documents](#), in §6.7.1
- [process the frame attributes](#), in §11.3.3
- [process the iframe attributes](#), in §4.7.6
- [process the resource appropriately](#), in §6.7.1
- [Process the response](#), in §7.1.3.2
- [product](#), in §7.6.1.1
- [profile](#)
 - [element-attr for head](#), in §11.2
 - [attribute for HTMLHeadElement](#), in §11.3.5
- [progress](#)
 - [event for media](#), in §4.7.14.15
 - [\(element\)](#), in §4.10.14
 - [event for global](#), in §Unnumbered section
- [proleptic Gregorian calendar](#), in §2.4.5
- [proleptic-Gregorian date](#), in §2.4.5
- [promise](#), in §7.1.3.9.2
- [PromiseRejectionEvent](#), in §7.1.3.9.2
- [PromiseRejectionEvent\(type, eventInitDict\)](#), in §7.1.3.9.2
- [prompt\(\)](#), in §7.5.1
- [prompt\(message\)](#), in §7.5.1
- [prompt\(message, default\)](#), in §7.5.1
- [prompt to unload](#), in §6.7.11
- [prompt to unload a document](#), in §6.7.11
- [Protected mode](#), in §5.7.2
- [protocol](#)
 - [attribute for HTMLHyperlinkElementUtils](#), in §4.8.3
 - [attribute for Location](#), in §6.6.4
- [proto-URL](#), in §7.6.1.3
- [provides a paint source](#), in §2.2.2
- [provides a stable state](#), in §7.1.4.2
- [provide such information](#), in §4.9.1.1

[push onto the list of active formatting elements](#), in §8.2.3.3
[pushState\(data, title\)](#), in §6.6.2
[pushState\(data, title, url\)](#), in §6.6.2
[q](#), in §4.5.7
[queryCommandEnabled\(\)](#), in §5.6.4
[queryCommandEnabled\(commandId\)](#), in §3.1.1
[queryCommandIndeterm\(\)](#), in §5.6.4
[queryCommandIndeterm\(commandId\)](#), in §3.1.1
[queryCommandState\(\)](#), in §5.6.4
[queryCommandState\(commandId\)](#), in §3.1.1
[queryCommandSupported\(\)](#), in §5.6.4
[queryCommandSupported\(commandId\)](#), in §3.1.1
[queryCommandValue\(\)](#), in §5.6.4
[queryCommandValue\(commandId\)](#), in §3.1.1
[queue a microtask](#), in §7.1.4.2
[queue a post-load task](#), in §11.3.4.2
[queue a progress post-load task](#), in §11.3.4.2
[queue a task](#), in §7.1.4.1
[queued](#), in §7.1.4.2
[queues a task](#), in §7.1.4.1
[queue that task as a post-load task](#), in §11.3.4.2
[queuing](#), in §7.1.4.1
[quirks mode](#), in §2.2.2
[radio](#)
 [attr-value for input/type](#), in §4.10.5
 [attr-value for menuitem/type](#), in §4.11.4
[Radio](#), in §4.11.4
[Radio Button](#), in §4.10.5.1.16
[radio button group](#), in §4.10.5.1.16
[radiogroup](#)
 [element-attr for menuitem](#), in §4.11.4
 [attribute for HTMLMenuItemElement](#), in §4.11.4
[RadioNodeList](#), in §2.7.2.2
[range](#), in §4.10.5
[Range](#), in §4.10.5.1.13
[rangeOverflow](#), in §4.10.21.3
[rangeUnderflow](#), in §4.10.21.3
[rank](#), in §4.3.6
[rarely necessary](#), in §4.10.19.7
[ratechange](#), in §4.7.14.15
[raw text](#), in §8.1.2
[raw text elements](#), in §8.1.2
[raw value](#), in §4.10.11
[rb](#), in §4.5.11
[read errors](#), in §2.2.2
[readiness state](#), in §4.7.14.11.1
[readonly](#)
 [element-attr for input](#), in §4.10.5.3.3
 [element-attr for textarea](#), in §4.10.11
[:read-only](#), in §4.15.2
[readOnly](#)
 [attribute for HTMLInputElement](#), in §4.10.5
 [attribute for HTMLTextAreaElement](#), in §4.10.11
[Read-only mode](#), in §5.7.2
[:read-write](#), in §4.15.2
[Read/write mode](#), in §5.7.2
[ready](#), in §4.7.14.11.1
[ready for post-load tasks](#), in §8.2.6
[readyState](#)
 [attribute for Document](#), in §3.1.2
 [attribute for HTMLTrackElement](#), in §4.7.13
 [attribute for HTMLMediaElement](#), in §4.7.14.7
[readystatechange](#), in §Unnumbered section
["ready to be parser-executed"](#), in §4.12.1.1
[realm execution context](#), in §7.1.3.1
[reason](#), in §7.1.3.9.2
[reassociateable](#), in §4.10.2
[Reassociateable element](#), in §4.10.2
[Reassociateable elements](#), in §4.10.2
[reconstruction of the active formatting element](#), in §8.2.3.3
[reconstruction of the active formatting elements](#), in §8.2.3.3
[reconstruct the active formatting element](#), in §8.2.3.3
[reconstruct the active formatting elements](#), in §8.2.3.3
[rect](#), in §4.7.16
[rectangle](#), in §4.7.16
[rectangle state](#), in §4.7.16
[Rectangle state](#), in §4.7.16
[rect state](#), in §4.7.16
[Rect state](#), in §4.7.16
[reentrant](#), in §8.2.1
[referrer](#), in §3.1.2
[referrer source](#), in §2.6.1
[reflect](#), in §2.7.1
[reflection](#), in §2.7.1
[refresh\(\)](#), in §7.6.1.5
[refresh](#), in §4.2.5.3
[refresh\(reload\)](#), in §7.6.1.5
[refused to allow the document to be unloaded](#), in §6.7.11
[refused to allow this document to be unloaded](#), in §6.7.11
[registerProtocolHandler\(scheme, url, title\)](#), in §7.6.1.3
[register the name](#), in §4.2.5.2
[register the names](#), in §4.2.5.2
[reinitialise url](#), in §4.8.3
[rel](#)
 [element-attr for link](#), in §4.2.4
 [attribute for HTMLLinkElement](#), in §4.2.4
 [attribute for HTMLAnchorElement](#), in §4.5.1
 [attribute for HTMLAreaElement](#), in §4.7.16
 [element-attr for a, area, links](#), in §4.8.2
[RelatedEvent](#), in §4.11.5.3
[RelatedEvent\(type\)](#), in §4.11.5.3
[RelatedEvent\(type, eventInitDict\)](#), in §4.11.5.3
[relatedTarget](#), in §4.11.5.3
[releaseEvents\(\)](#)
 [method for Document](#), in §11.3.5
 [method for Window](#), in §11.3.5
[Relevant](#), in §7.1.3.5
[relevant application cache](#), in §11.3.4
[relevant child nodes](#), in §9.3
[relevant Document](#), in §6.6.4
[relevant global object](#), in §7.1.3.5.4
[relevant mutations](#), in §4.7.5
[relevant Realm](#), in §7.1.3.5.4
[relevant settings object](#), in §7.1.3.5.4

relList

[attribute for HTMLLinkElement](#), in §4.2.4

[attribute for HTMLAnchorElement](#), in §4.5.1

[attribute for HTMLAreaElement](#), in §4.7.16

[reload\(\)](#), in §6.6.4

[reload override buffer](#), in §3.1

[reload override flag](#), in §3.1

[reload-triggered navigation](#), in §6.7.1

[removeCue\(cue\)](#), in §4.7.14.11.5

[removed from](#), in §2.1.3

[removed from a document](#), in §2.1.3

remove(index)

[method for HTMLOptionsCollection](#), in §2.7.2.3

[method for HTMLSelectElement](#), in §4.10.7

[method for DataTransferItemList](#), in §5.7.3.1

[removetrack](#), in §4.7.14.15

[rendered legend](#), in §10.3.13

[reparsed](#), in §2.5.2

[replace\(\)](#), in §6.6.4

[replaced element](#), in §2.2.2

[replacement enabled](#), in §6.7.10

[replacement must be enabled](#), in §6.7.10

[replaceState\(data, title\)](#), in §6.6.2

[replaceState\(data, title, url\)](#), in §6.6.2

[replace\(url\)](#), in §6.6.4

[report an error](#), in §7.1.3.8

[report an exception](#), in §7.1.3.8.1

[reported MIME types](#), in §7.6.1.5

[report the error](#), in §7.1.3.8

[report the exception](#), in §7.1.3.8.1

reportValidity()

[method for HTMLFormElement](#), in §4.10.3

[method for HTMLObjectElement](#), [HTMLInputElement](#), [HTMLButtonElement](#), [HTMLSelectElement](#), [HTMLTextAreaElement](#), [HTMLKeygenElement](#), [HTMLOutputElement](#), [HTMLFieldSetElement](#), in §4.10.21.3

[represent](#), in §3.2.2

[reprocess the iframe attributes](#), in §4.7.6

[requestAnimationFrame\(\)](#), in §7.8

[requestAnimationFrame\(callback\)](#), in §7.8

[:required](#), in §4.15.2

required

[attribute for HTMLInputElement](#), in §4.10.5

[element-attr for input](#), in §4.10.5.3.4 [definition of](#), in §4.10.5.3.4

[element-attr for select](#), in §4.10.7

[attribute for HTMLSelectElement](#), in §4.10.7

[element-attr for textarea](#), in §4.10.11

[attribute for HTMLTextAreaElement](#), in §4.10.11

reset

[attr-value for input/type](#), in §4.10.5

[attr-value for button/type](#), in §4.10.6

[definition of](#), in §4.10.23

[event for global](#), in §Unnumbered section

[reset\(\)](#), in §4.10.3

[reset algorithm](#), in §4.10.23

[reset button](#), in §4.10.6

[Reset Button](#), in §4.10.5.1.20

[resettable](#), in §4.10.2

[Resettable element](#), in §4.10.2

[Resettable elements](#), in §4.10.2

[reset the form owner](#), in §4.10.18.3

[reset the insertion mode appropriately](#), in §8.2.3.1

resource

[dfn for http](#), in §2.1.1

[definition of](#), in §4.7.14

[resource fetch algorithm](#), in §4.7.14.5

[resource selection algorithm](#), in §4.7.14.5

[responsible browsing context](#), in §7.1.3.1

[responsible document](#), in §7.1.3.1

[responsible event loop](#), in §7.1.3.1

[restart the animation](#), in §10.4.2

[restore persisted user state](#), in §6.7.10.1

[restrictions](#), in §8.1.2.6

[restrictions on the contents of raw text and escapable raw text elements](#), in §8.1.2.6

[resulting URL record](#), in §2.5.2

[resulting URL records](#), in §2.5.2

[resulting URL string](#), in §2.5.2

returnValue

[attribute for BeforeUnloadEvent](#), in §6.7.11.1

[attribute for WindowModal](#), in §7.5.3

[return value](#), in §7.5.3

[return value origin](#), in §7.5.3

rev

[element-attr for a, link, links](#), in §4.2.4

[attribute for HTMLLinkElement](#), in §4.2.4

[attribute for HTMLAnchorElement](#), in §4.5.1

reversed

[element-attr for ol](#), in §4.4.5

[attribute for HTMLOListElement](#), in §4.4.5

[reverse link](#), in §4.2.4

[Reverse links](#), in §4.2.4

right

[attr-value for marquee/direction](#), in §10.5.11

[state for marquee](#), in §11.3.2

[rightmargin](#), in §11.2

[role](#), in §2.2.2

[root element](#), in §2.1.3

[root element of a Document object](#), in §2.1.3

[root elements](#), in §2.1.3

row

[attr-value for scope](#), in §4.9.10

[state for scope](#), in §4.9.10

[definition of](#), in §4.9.12

[rowgroup](#), in §4.9.10

row group

[state for scope](#), in §4.9.10

[definition of](#), in §4.9.12

[row group header](#), in §4.9.12.2

[row groups](#), in §4.9.12

[row header](#), in §4.9.12.2

[rowIndex](#), in §4.9.8

rows

[attribute for HTMLTableElement](#), in §4.9.1

[attribute for HTMLTableSectionElement](#), in §4.9.5

[definition of](#), in §4.9.12

[element-attr for textarea](#), in §4.10.11

[attribute for HTMLTextAreaElement](#), in §4.10.11

[element-attr for frameset](#), in §11.3.3

[rowSpan](#), in §4.9.11

[rowspan](#), in §4.9.11

[rp](#), in §4.5.14

[rt](#), in §4.5.12

[rtc](#), in §4.5.13

rtl

[attr-value for global/dir](#), in §3.2.5.6

[state for dir](#), in §3.2.5.6

[ruby](#), in §4.5.10

[ruby annotation container](#), in §4.5.10

[ruby base container](#), in §4.5.10

[ruby bases](#), in §4.5.10

[ruby segment](#), in §4.5.10

[ruby text annotations](#), in §4.5.10

[ruby text container](#), in §4.5.10

rules

[element-attr for table](#), in §11.2

[attribute for HTMLTableElement](#), in §11.3.5

[rules for constructing the chapter tree from a text track](#), in §4.7.14.11.7

[rules for distinguishing if a resource is text or binary](#), in §2.6.4

[rules for extracting the chapter title](#), in §4.7.14.11.1

[rules for interpreting WebVTT cue text](#), in §2.2.2

[rules for parsing a hash-name reference](#), in §2.4.9

[rules for parsing a legacy color value](#), in §2.4.6

[rules for parsing a legacy font size](#), in §10.3.4

[rules for parsing a list of dimensions](#), in §2.4.4.7

[rules for parsing a list of floating-point numbers](#), in §2.4.4.6

[rules for parsing dimension values](#), in §2.4.4.4

[rules for parsing floating-point number values](#), in §2.4.4.3

[rules for parsing integer](#), in §2.4.4.1

[rules for parsing integers](#), in §2.4.4.1

[rules for parsing manifests](#), in §11.3.4.1

[rules for parsing non-negative integers](#), in §2.4.4.2

[rules for parsing non-zero dimension values](#), in §2.4.4.5

[rules for parsing signed integers](#), in §2.4.4.1

[rules for parsing simple color values](#), in §2.4.6

[rules for serializing simple color values](#), in §2.4.6

[rules for sniffing images specifically](#), in §2.6.4

[rules for updating the display of WebVTT text tracks](#), in §2.2.2

[rules for updating the text track rendering](#), in §4.7.14.11.1

[run a classic script](#), in §7.1.3.4

[run authentic click activation steps](#), in §5.3

[run canceled activation steps](#), in §5.3

[run CSS animations and send events](#), in §7.1.4.2

[running script](#), in §7.1.3.4

[running synthetic click activation steps](#), in §5.3

[running the classic script](#), in §7.1.3.4

[run post-click activation steps](#), in §5.3

[run pre-click activation steps](#), in §5.3

[runs](#), in §4.7.14.5

[run synthetic click activation steps](#), in §5.3

[run the animation frame callbacks](#), in §7.8

[run the classic script](#), in §7.1.3.4

[run the fullscreen rendering steps](#), in §7.1.4.2

[run the global script clean-up jobs](#), in §7.1.3.4

[Runtime script errors](#), in §7.1.3.8

[s](#), in §4.5.5

[safelisted schemes](#), in §7.6.1.3

[salvageable](#), in §6.7.11

[same origin](#), in §6.4

[same origin-domain](#), in §6.4

[samp](#), in §4.5.19

sandbox

[element-attr for iframe](#), in §4.7.6

[attribute for HTMLIFrameElement](#), in §4.7.6

[sandbox cookies](#), in §3.1.2

[sandboxed automatic features browsing context flag](#), in §6.5

[sandboxed auxiliary navigation browsing context flag](#), in §6.5

[sandboxed document.domain browsing context flag](#), in §6.5

[sandboxed forms browsing context flag](#), in §6.5

[sandboxed fullscreen browsing context flag](#), in §6.5

[sandboxed into a unique origin](#), in §6.5

[sandboxed modals flag](#), in §6.5

[sandboxed navigation browsing context flag](#), in §6.5

[sandboxed origin browsing context flag](#), in §6.5

[sandboxed plugins browsing context flag](#), in §6.5

[sandboxed pointer lock browsing context flag](#), in §6.5

[sandboxed scripts browsing context flag](#), in §6.5

[sandboxed storage area URLs flag](#), in §6.5

[sandboxed top-level navigation browsing context flag](#), in §6.5

[sandboxing flag set](#), in §6.5

[sandbox propagates to auxiliary browsing contexts flag](#), in §6.5

[satisfies its constraints](#), in §4.10.21.1

[satisfy its constraints](#), in §4.10.21.1

[satisfy their constraints](#), in §4.10.21.1

scheme

[element-attr for meta](#), in §11.2

[attribute for HTMLMetaElement](#), in §11.3.5

scope

[element-attr for th](#), in §4.9.10

[attribute for HTMLTableHeaderCellElement](#), in §4.9.10

[element-attr for td](#), in §11.2

script

[\(element\)](#), in §4.12.1

[dfn for concept](#), in §7.1.3.1

[script-closable](#), in §6.3.1

[script content restrictions](#), in §4.12.1.3

[script-created parser](#), in §7.3.1

[script documentation](#), in §4.12.1.4

[scripting](#), in §7.1.3.1

[scripting flag](#), in §8.2.3.5

[Scripting is disabled](#), in §7.1.2

[Scripting is disabled for a node](#), in §7.1.2

[scripting is enabled](#), in §7.1.2

[Scripting is enabled for a node](#), in §7.1.2

[scripting was enabled](#), in §7.1.2

[scripting was enabled or not](#), in §8.2.3.5

[script nesting level](#), in §8.2.1

scripts

[attribute for Document](#), in §3.1.3

[dfn for concept](#), in §7.1.3.1

[Script-supporting element](#), in §3.2.4.2.9

[Script-supporting elements](#), in §3.2.4.2.9

scroll

[attr-value for marquee/behavior](#), in §11.3.2

[state for marquee/behavior](#), in §11.3.2

[scrollamount](#), in §11.3.2

[scrollAmount](#), in §11.3.2

[scrollbars](#), in §6.3.6

[scrolldelay](#), in §11.3.2

[scrollDelay](#), in §11.3.2

scrolling

[element-attr for iframe](#), in §11.2

[attribute for HTMLFrameElement](#), in §11.3.3

[attribute for HTMLIFrameElement](#), in §11.3.5

[scrollRestoration](#), in §6.6.2

[scroll restoration mode](#), in §6.6.1

[scroll to the fragment identifier](#), in §6.7.9

[Search](#), in §4.10.5.1.2

search

[attribute for HTMLHyperlinkElementUtils](#), in §4.8.3

[attr-value for link/type](#), in §4.8.6.9

[attr-value for input/type](#), in §4.10.5

[attribute for Location](#), in §6.6.4

[second administrative level](#), in §4.10.19.8.1

section

[\(element\)](#), in §4.3.3

[definition of](#), in §4.3.10.1

[sectioning content](#), in §3.2.4.2.3

[sectioning roots](#), in §4.3.10

[sectionRowIndex](#), in §4.9.8

[secured](#), in §2.1.5

[Sec-WebSocket-Protocol](#), in §2.2.2

[seek](#), in §4.7.14.9

[seekable](#), in §4.7.14.9

[seeked](#), in §4.7.14.15

seeking

[attribute for HTMLMediaElement](#), in §4.7.14.9

[event for media](#), in §4.7.14.15

[segmentation and categorization of content of a ruby](#), in §4.5.10

["select"](#), in §4.10.20

select

[\(element\)](#), in §4.10.7

[enum-value for SelectionMode](#), in §4.10.20

[enum for SelectionMode](#), in §4.10.20

[event for global](#), in §Unnumbered section

[select\(\)](#), in §4.10.20

[select an application cache](#), in §11.3.4

[select an image source](#), in §4.7.5

selected

[attribute for VideoTrack](#), in §4.7.14.10.1

[element-attr for option](#), in §4.10.10

[attribute for HTMLOptionElement](#), in §4.10.10

[selected coordinate](#), in §4.10.5.1.19

[selected files](#), in §4.10.5.1.17

selectedIndex

[attribute for HTMLOptionsCollection](#), in §2.7.2.3

[attribute for VideoTrackList](#), in §4.7.14.10.1

[attribute for HTMLSelectElement](#), in §4.10.7

[selectedness](#), in §4.10.10

[selectedOptions](#), in §4.10.7

[selecting an image source](#), in §4.7.5

[selectionDirection](#), in §4.10.20

[selectionEnd](#), in §4.10.20

[selectionStart](#), in §4.10.20

[self](#), in §6.3

[self-closing flag](#), in §8.2.4

[send a signal](#), in §11.3.4.8

[send a WebSocket Message](#), in §2.2.2

[sends a signal](#), in §11.3.4.8

[send select update notifications](#), in §4.10.7

[Separators](#), in §4.11.3

[sequential focus navigation](#), in §5.4.5

[sequential focus navigation order](#), in §5.4.5

[sequential focus navigation starting point](#), in §5.4.5

[sequential navigation search algorithm](#), in §5.4.5

[session history](#), in §6.6.1

[session history document visibility change steps](#), in §6.7.10

[session history entry](#), in §6.6.1

[session history event loop](#), in §6.6.2

[session history traversal](#), in §6.7.10

[session history traversal queue](#), in §6.6.2

[set-cookie](#), in §4.2.5.3

[setCustomValidity\(\)](#), in §4.10.21.3

[setCustomValidity\(error\)](#), in §4.10.21.3

[setData\(format, data\)](#), in §5.7.3

[setDragImage\(image, x, y\)](#), in §5.7.3

[setInterval\(handler, timeout, arguments...\)](#), in §7.4

[setInterval\(handler, timeout, arguments\)](#), in §7.4

[set of comma-separated tokens](#), in §2.4.8

[set of scripts that will execute as soon as possible](#), in §4.12.1.1

[set of space-separated tokens](#), in §2.4.7

[setRangeText\(\)](#), in §4.10.20

[setRangeText\(replacement\)](#), in §4.10.20

[setRangeText\(replacement, start, end\)](#), in §4.10.20

[setRangeText\(replacement, start, end, selectionMode\)](#), in §4.10.20

[setSelectionRange\(\)](#), in §4.10.20

[setSelectionRange\(start, end\)](#), in §4.10.20

[setSelectionRange\(start, end, direction\)](#), in §4.10.20

[set the document's address](#), in §6.7.1

[set the frozen base URL](#), in §4.2.3

[set the url](#), in §4.8.3

[set the value of a new indexed property](#), in §4.10.7

[set the value of a new indexed property or set the value of an existing indexed property](#), in §2.7.2.3

[setTimeout\(handler, timeout, arguments\)](#), in §7.4

[setTimeout\(handler, timeout, arguments...\)](#), in §7.4

[setting](#), in §2.1.4

[settings object](#), in §7.1.3.1

[setting the document's address](#), in §6.7.1

[set up a browsing context environment settings object](#), in §6.1.6

[Set up the request](#), in §7.1.3.2

[shape](#)

- [element-attr for area](#), in §4.7.16
- [attribute for HTMLAreaElement](#), in §4.7.16
- [element-attr for a](#), in §11.2
- [attribute for HTMLAnchorElement](#), in §11.3.5

[Shift JIS](#), in §8.2.2.3

[shipping](#), in §4.10.19.8.1

[should be used](#), in §4.7.8

[show](#), in §Unnumbered section

[showing](#)

- [mode for track](#), in §4.7.14.11.1
- [enum-value for TextTrackMode](#), in §4.7.14.11.5
- [enum for TextTrackMode](#), in §4.7.14.11.5

["showing"](#), in §4.7.14.11.5

[showModalDialog\(\)](#), in §7.5.3

[showModalDialog\(url\)](#), in §7.5.3

[showModalDialog\(url, argument\)](#), in §7.5.3

[shown](#), in §2.1

[show poster flag](#), in §4.7.14.6

[shows caching progress](#), in §11.3.4.2

[sign](#), in §4.7.14.10.1

[signal a type change](#), in §4.10.5

[simple color](#), in §2.4.6

[size](#)

- [definition of](#), in §2.6.2
- [attribute for HTMLInputElement](#), in §4.10.5
- [element-attr for input](#), in §4.10.5.3.2
- [element-attr for select](#), in §4.10.7
- [attribute for HTMLSelectElement](#), in §4.10.7
- [element-attr for hr](#), in §11.2
- [attribute for HTMLFontElement](#), in §11.3.5
- [attribute for HTMLHRElement](#), in §11.3.5

[sizes](#)

- [attribute for HTMLLinkElement](#), in §4.2.4
- [element-attr for source](#), in §4.7.4
- [element-attr for img](#), in §4.7.5
- [attribute for HTMLImageElement](#), in §4.7.5
- [element-attr for link](#), in §4.8.6.5

[skip whitespace](#), in §2.4.1

[slide](#)

- [attr-value for marquee/behavior](#), in §11.3.2
- [state for marquee/behavior](#), in §11.3.2

[slots](#), in §4.9.12

[small](#), in §4.5.4

[sms:](#), in §2.2.2

[Soft](#), in §4.10.11

[soft](#), in §4.10.11

[solitary callback microtasks](#), in §7.1.4.2

[source](#), in §4.7.12

[source browsing context](#), in §6.7.1

[source node](#), in §5.7.5

[source set](#), in §4.7.5

[source size](#), in §4.7.5

[Source text](#), in §7.1.3.1

[space characters](#), in §2.4.1

[spacer](#), in §11.2

[span](#)

- [\(element\)](#), in §4.5.28
- [element-attr for colgroup](#), in §4.9.3
- [attribute for HTMLTableColElement](#), in §4.9.3
- [element-attr for col](#), in §4.9.4

[span multiple columns](#), in §4.9.11

[Special](#), in §8.2.3.2

[specified](#), in §5.7.8

[specifies an operation](#), in §5.7.8

[specify an operation](#), in §5.7.8

[spellcheck](#), in §5.6.5

[spinning the event loop](#), in §7.1.4.2

[spins the event loop](#), in §7.1.4.2

[spin the event loop](#), in §7.1.4.2

[split a string on commas](#), in §2.4.8

[split a string on spaces](#), in §2.4.7

[spoon-feed the parser](#), in §9.2

src	start	stop()
element-attr for img , in §4.7.5	element-attr for ol , in §4.4.5	method for Window , in §6.3.1
attribute for HTMLImageElement , in §4.7.5	attribute for HTMLListElement , in §4.4.5	method for HTMLMarqueeElement , in §11.3.2
element-attr for iframe , in §4.7.6	enum-value for SelectionMode , in §4.10.20	stopped , in §8.2.6
attribute for HTMLIFrameElement , in §4.7.6	enum for SelectionMode , in §4.10.20	stopped due to errors , in §4.7.14.8
element-attr for embed , in §4.7.7	event for marquee , in §11.3.2	stopped parsing , in §8.2.6
attribute for HTMLEmbedElement , in §4.7.7	"start" , in §4.10.20	stops parsing , in §8.2.6
element-attr for source , in §4.7.12	start(index) , in §4.7.14.13	storage , in §Unnumbered section
attribute for source , in §4.7.12	start tag , in §8.1.2.1	strictly split , in §2.4.1
element-attr for track , in §4.7.13	start tags , in §8.1.2.1	strictly split a string , in §2.4.1
attribute for HTMLTrackElement , in §4.7.13	start the track processing model , in §4.7.14.11.3	strictly splitting the string , in §2.4.1
element-attr for media , in §4.7.14.2	start the WebSocket closing handshake , in §2.2.2	strike , in §11.2
attribute for HTMLMediaElement , in §4.7.14.2	startTime , in §4.7.14.11.5	stringification behavior
attribute for HTMLInputElement , in §4.10.5	state	dfn for HTMLHyperlinkElementUtils , in §4.8.3
element-attr for input , in §4.10.5.1.19	dfn for image , in §4.7.5	dfn for Location , in §6.6.4
element-attr for script , in §4.12.1	attribute for History , in §6.6.2	strip and collapse whitespace , in §2.4.1
attribute for HTMLScriptElement , in §4.12.1	attribute for PopStateEvent , in §6.7.10.2	strip leading and trailing whitespace , in §2.4.1
element-attr for frame , in §11.3.3	state object , in §6.6.1	strip line breaks , in §2.4.1
attribute for HTMLFrameElement , in §11.3.3	state of the type attribute , in §4.10.5.1	stripped line breaks , in §2.4.1
srcdoc	states of the type attribute , in §4.10.5.1	stripping and collapsing whitespace , in §2.4.1
element-attr for iframe , in §4.7.6	statically validate the constraints , in §4.10.21.2	stripping leading and trailing whitespace , in §2.4.1
attribute for HTMLIFrameElement , in §4.7.6	statically validating the constraints , in §4.10.21.2	strong , in §4.5.3
srclang	status	StructuredClone , in §2.9.4
element-attr for track , in §4.7.13	attribute for Window , in §6.3.6	StructuredCloneWithTransfer , in §2.9.3
attribute for HTMLTrackElement , in §4.7.13	attribute for ApplicationCache , in §11.3.4.8	style
srcObject , in §4.7.14.2	statusbar , in §6.3.6	element-attr for global , in §3.2.5.8
srcset	step	(element) , in §4.2.6
element-attr for source , in §4.7.4	attribute for HTMLInputElement , in §4.10.5	style data , in §4.2.6
element-attr for img , in §4.7.5	element-attr for input , in §4.10.5.3.8	stylesheet , in §4.8.6.10
attribute for HTMLImageElement , in §4.7.5	step-align , in §4.10.5.4	style sheet ready , in §4.2.7
stack of open elements , in §8.2.3.2	step-aligned , in §4.10.5.4	sub , in §4.5.21
stack of template insertion modes , in §8.2.3.1	step-aligns , in §4.10.5.4	submenu label , in §4.11.3
stalled , in §4.7.14.15	step base , in §4.10.5.3.8	submit() , in §4.10.3
stall timeout , in §4.7.14.5	stepDown(n) , in §4.10.5.4	submit
standby	stepMismatch , in §4.10.21.3	attr-value for input/type , in §4.10.5
element-attr for object , in §11.2	step scale factor , in §4.10.5.3.8	attr-value for button/type , in §4.10.6
attribute for HTMLObjectElement , in §11.3.5	steps to expose a media-resource-specific text track , in §4.7.14.11.2	definition of , in §4.10.22.3
start() , in §11.3.2	stepUp(n) , in §4.10.5.4	event for global , in §Unnumbered section
		Submit as entity body , in §4.10.22.3

[submit button](#)

[definition of](#), in §4.10.2

[element-state for button/type](#), in §4.10.6

[Submit Button](#), in §4.10.5.1.18

[submit buttons](#), in §4.10.2

[submittable](#), in §4.10.2

[Submittable element](#), in §4.10.2

[Submittable elements](#), in §4.10.2

[submitted](#), in §4.10.22.3

[subprotocol in use](#), in §2.2.2

[substantial](#), in §10.3.10

[Subtitles](#), in §4.7.13

[subtitles](#)

[attr-value for track/kind](#), in §4.7.13

[attr-value for commonTrack/kind](#), in §4.7.14.10.1

[definition of](#), in §4.7.14.11.1

[enum-value for TextTrackKind](#), in §4.7.14.11.5

["subtitles"](#), in §4.7.14.11.5

[suffer from a custom error](#), in §4.10.21.1

[suffer from an overflow](#), in §4.10.21.1

[suffer from an underflow](#), in §4.10.21.1

[suffer from a pattern mismatch](#), in §4.10.21.1

[suffer from a step mismatch](#), in §4.10.21.1

[suffer from a type mismatch](#), in §4.10.21.1

[suffer from bad input](#), in §4.10.21.1

[suffer from being missing](#), in §4.10.21.1

[suffer from being too long](#), in §4.10.21.1

[suffer from being too short](#), in §4.10.21.1

[suffering from a custom error](#), in §4.10.21.1

[suffering from an overflow](#), in §4.10.21.1

[suffering from an underflow](#), in §4.10.21.1

[suffering from a pattern mismatch](#), in §4.10.21.1

[suffering from a step mismatch](#), in §4.10.21.1

[suffering from a type mismatch](#), in §4.10.21.1

[suffering from bad input](#), in §4.10.21.1

[suffering from being missing](#), in §4.10.21.1

[suffering from being too long](#), in §4.10.21.1

[suffering from being too short](#), in §4.10.21.1

[suffixes](#), in §7.6.1.5

[suggestions source element](#), in §4.10.5.3.9

[suitable sequentially focusable area](#), in §5.4.5

[summary](#)

[\(element\)](#), in §4.11.2

[element-attr for table](#), in §11.2

[attribute for HTMLTableElement](#), in §11.3.5

[sup](#), in §4.5.21

[supported](#), in §2.1.1

[supported token](#), in §2.2.2

[supported tokens](#), in §2.2.2

[supporting the suggested default rendering](#), in §2.2.1

[support the suggested default rendering](#), in §2.2.1

[suspend](#), in §4.7.14.15

[svg](#), in §4.7.19

[SVG namespace](#), in §2.8

[swapCache\(\)](#), in §11.3.4.8

[synchronous section](#), in §7.1.4.2

[synchronous sections](#), in §7.1.4.2

[syntax for which](#), in §8.1.2.3

[tabindex](#), in §5.4.3

[tabIndex](#), in §5.4.3

[tabindex focus flag](#), in §5.4.3

[table](#)

[\(element\)](#), in §4.9.1

[definition of](#), in §4.9.12

[table design techniques](#), in §4.9.1.1

[table model](#), in §4.9.12

[table model error](#), in §4.9.12

[tables](#), in §4.9.12

[tag](#), in §4.8.6.11

[Tag clouds](#), in §4.13.3

[tag name](#), in §8.1.2

[Tag omission in text/html](#), in §3.2.3

[Tags](#), in §8.1.2

[taintEnabled\(\)](#), in §7.6.1.1

[:target](#), in §6.7.9

[target](#)

[element-attr for base](#), in §4.2.3

[attribute for HTMLBaseElement](#), in §4.2.3

[attribute for HTMLAnchorElement](#), in §4.5.1

[attribute for HTMLAreaElement](#), in §4.7.16

[element-attr for a, area, links](#), in §4.8.2

[element-attr for form](#), in §4.10.19.6

[attribute for HTMLFormElement](#), in §4.10.19.6

[element-attr for link](#), in §11.2

[attribute for HTMLLinkElement](#), in §11.3.5

[target element](#), in §6.7.9

[target override](#), in §2.2.2

[task queues](#), in §7.1.4.1

[tasks](#), in §7.1.4.1

[task source](#), in §7.1.4.1

[tBodies](#), in §4.9.1

[tbody](#), in §4.9.5

[td](#), in §4.9.9

[tel](#)

[attr-value for input/type](#), in §4.10.5

[value for form/inputmode](#), in §4.10.19.7

[Telephone](#)

[element-state for input](#), in §4.10.5.1.3

[state for inputmode](#), in §4.10.19.7

[template](#), in §4.12.3

[template contents](#), in §4.12.3

[temporary buffer](#), in §8.2.4

[term](#), in §4.5.8

[termination nesting level](#), in §6.7.11

[Text](#)

[definition of](#), in §3.2.4.2.5

[element-state for input](#), in §4.10.5.1.2

text

[attribute for HTMLTitleElement](#), in §4.2.2

[attribute for HTMLAnchorElement](#), in §4.5.1

[attr-value for input/type](#), in §4.10.5

[attribute for HTMLOptionElement](#), in §4.10.10

[attribute for HTMLScriptElement](#), in §4.12.1

[element-attr for body](#), in §11.2

[attribute for HTMLBodyElement](#), in §11.3.5

[textarea](#), in §4.10.11

[textarea effective height](#), in §10.5.15

[textarea effective width](#), in §10.5.15

[textarea wrapping transformation](#), in §4.10.11

[Text content](#), in §3.2.4.2.5

[textLength](#), in §4.10.11

[text/plain](#), in §4.10.19.6

[text/plain encoding algorithm](#), in §4.10.22.8

[TextTrack](#), in §4.7.14.11.5

[text track cue](#), in §4.7.14.11.1

[TextTrackCue](#), in §4.7.14.11.5

[text track cue active flag](#), in §4.7.14.11.1

[text track cue data](#), in §4.7.14.11.1

[text track cue display state](#), in §4.7.14.11.1

[text track cue end time](#), in §4.7.14.11.1

[text track cue identifier](#), in §4.7.14.11.1

[TextTrackCueList](#), in §4.7.14.11.5

[text track cue order](#), in §4.7.14.11.1

[text track cue pause-on-exit flag](#), in §4.7.14.11.1

[text track cue start time](#), in §4.7.14.11.1

[text track cue writing direction](#), in §2.2.2

[text track failed to load](#), in §4.7.14.11.1

[text track in-band metadata track dispatch type](#), in §4.7.14.11.1

[text track kind](#), in §4.7.14.11.1

[text track kinds](#), in §4.7.14.11.1

[text track label](#), in §4.7.14.11.1

[text track language](#), in §4.7.14.11.1

[TextTrackList](#), in §4.7.14.11.5

[text track list of cues](#), in §4.7.14.11.1

[text track loaded](#), in §4.7.14.11.1

[text track loading](#), in §4.7.14.11.1

[text track mode](#), in §4.7.14.11.1

[text track not loaded](#), in §4.7.14.11.1

[text track readiness state](#), in §4.7.14.11.1

[text track rules for extracting the chapter title](#), in §4.7.14.11.1

[textTracks](#), in §4.7.14.11.5

[text tracks](#), in §4.7.14.11.1

[tfoot](#), in §4.9.1

[tfoot](#), in §4.9.7

[th](#), in §4.9.10

[that algorithm](#), in §8.4

[thead](#), in §4.9.6

[thead](#), in §4.9.1

[the address](#), in §3.1

[the conditions described above](#), in §4.7.5.1.16

[the directionality](#), in §3.2.5.6

[the document address](#), in §3.1

[the Document object is indexed for property retrieval](#), in §3.1.3

[the document's address](#), in §3.1

[The document's referrer](#), in §3.1

[The drag data item kind](#), in §5.7.2

[The drag data item type string](#), in §5.7.2

[the element's directionality](#), in §3.2.5.6

[The embed element setup steps](#), in §4.7.7

[the empty string](#), in §4.7.14.3

[the environment settings object's global object](#), in §7.1.3.5

[the environment settings object's Realm](#), in §7.1.3.5

[The event handler processing algorithm](#), in §7.1.5.1

[the first 1024 bytes](#), in §4.2.5.5

[the global object's Realm](#), in §7.1.3.5

[the handling for misnested tags](#), in §8.2.5.4.7

[The HTML syntax](#), in §8

[the indicated part of the document](#), in §6.7.9

[the kind of text track](#), in §4.7.14.11.1

[the link is an alternative stylesheet](#), in §4.8.6.10

[The location bar BarProp object](#), in §6.3.6

[the manifest](#), in §11.3.4

[The menu bar BarProp object](#), in §6.3.6

[the outline's owner](#), in §4.3.10.1

[The personal bar BarProp object](#), in §6.3.6

[the Realm's global object](#), in §7.1.3.5

[the Realm's settings object](#), in §7.1.3.5

[the resource's content-type metadata](#), in §2.6.4

[the rules described previously](#), in §4.7.14.5

[The rules for choosing a browsing context given a browsing context name](#), in §6.1.5

[the script is ready](#), in §4.12.1.1

[the script's script](#), in §4.12.1.1

[the script's type](#), in §4.12.1.1

[The scrollbar BarProp object](#), in §6.3.6

[The status bar BarProp object](#), in §6.3.6

[the step labeled fragment identifiers](#), in §6.7.1

[the text insertion mode](#), in §8.2.5.4.8

[the text tracks are ready](#), in §4.7.14.11.1

[The toolbar BarProp object](#), in §6.3.6

[the WebSocket closing handshake is started](#), in §2.2.2

[the WebSocket connection close code](#), in §2.2.2

[the WebSocket connection close reason](#), in §2.2.2

[the WebSocket connection is closed](#), in §2.2.2

[the WebSocket connection is established](#), in §2.2.2

[the Window object is indexed for property retrieval](#), in §6.3.3

[The XHTML syntax](#), in §9

[third administrative level](#), in §4.10.19.8.1

[this](#)

[dfn for conventions](#), in §1.7.2

[element for conventions](#), in §1.7.2

[through which new document is nested](#), in §6.1.1

[throw](#), in §2.2.2

[time](#)

[definition of](#), in §2.4.5.4

[\(element\)](#), in §4.5.16

[attr-value for input/type](#), in §4.10.5

[Time](#), in §4.10.5.1.10

[timeline offset](#), in §4.7.14.6

[time marches on](#), in §4.7.14.8

[TimeRanges](#), in §4.7.14.13

[timer initialization steps](#), in §7.4

[timer nesting level](#), in §7.4

[timer task source](#), in §7.4

[timeupdate](#), in §4.7.14.15

[time zone](#), in §2.4.5.6

[time-zone offset](#), in §2.4.5.6

[title](#)

[attribute for Document](#), in §3.1.3

[element-attr for global, figure, div, img, textarea, meter](#), in §3.2.5.2

[\(element\)](#), in §4.2.2

[element-attr for link](#), in §4.2.4

[element-attr for style](#), in §4.2.6

[element-attr for dfn](#), in §4.5.8

[element-attr for abbr](#), in §4.5.9

[element-attr for input](#), in §4.10.5.3.6

[element-attr for menuitem](#), in §4.11.4

[toBlob\(\)](#), in §4.12.4

[toBlob\(callback\)](#), in §4.12.4

[toBlob\(callback, type\)](#), in §4.12.4

[toBlob\(callback, type, arguments...\)](#), in §4.12.4

[toDataURL\(\)](#), in §4.12.4

[toggle](#)

[definition of](#), in §4.10.7

[event for global](#), in §Unnumbered section

[tokenization](#), in §8.2.4

[tokenizer](#), in §8.2.4

[toolbar](#), in §6.3.6

[tooLong](#), in §4.10.21.3

[tooShort](#), in §4.10.21.3

[top](#)

[definition of](#), in §6.1.1.1

[attribute for Window](#), in §6.3

[top-level browsing context](#), in §6.1.1

[@@toPrimitive](#), in §2.2.2

[@@toStringTag](#), in §2.2.2

[tr](#), in §4.9.8

[track](#)

[\(element\)](#), in §4.7.13

[attribute for HTMLTrackElement](#), in §4.7.13

[attribute for TextTrackCue](#), in §4.7.14.11.5

[attribute for TrackEvent](#), in §4.7.14.14

[TrackEvent](#), in §4.7.14.14

[TrackEvent\(type\)](#), in §4.7.14.14

[TrackEvent\(type, eventInitDict\)](#), in §4.7.14.14

[track label](#), in §4.7.13

[track language](#), in §4.7.13

[track URL](#), in §4.7.13

[Transfer](#), in §2.9.2

[Transferable objects](#), in §2.9.2

[TransferHelper](#), in §2.9.6

[transformToDocument\(\)](#), in §2.2.2

[transformToFragment\(\)](#), in §2.2.2

[translatable attributes](#), in §3.2.5.4

[translate](#), in §3.2.5.4

[translate-enabled](#), in §3.2.5.4

[translation](#), in §4.7.14.10.1

[translation mode](#), in §3.2.5.4

[transparent](#), in §3.2.4.3

[transparently follow the redirect](#), in §2.6.2

[traverse the history](#), in §6.7.10

[traverse the history by a delta](#), in §6.6.2

[traversing the history](#), in §6.7.10

[tree construction dispatcher](#), in §8.2.5

[tree order](#), in §2.1.3

[true-by-default](#), in §5.6.5

[truespeed](#), in §11.3.2

[trueSpeed](#), in §11.3.2

[trusted](#), in §2.1.4

[trusted event](#), in §2.1.4

[tt](#), in §11.2

[tuple](#), in §6.4

[tuple origin](#), in §6.4

[turned off](#), in §11.3.2

[turned on](#), in §11.3.2

type

[element-attr for link](#), in §4.2.4

[attribute for HTMLLinkElement](#), in §4.2.4

[element-attr for style](#), in §4.2.6

[attribute for HTMLStyleElement](#), in §4.2.6

[element-attr for ol](#), in §4.4.5

[attribute for HTMLListElement](#), in §4.4.5

[attribute for HTMLAnchorElement](#), in §4.5.1

[element-attr for embed](#), in §4.7.7

[element-attr for object](#), in §4.7.8

[element-attr for source](#), in §4.7.12

[attribute for source](#), in §4.7.12

[attribute for HTMLAreaElement](#), in §4.7.16

[element-attr for a, links](#), in §4.8.2

[element-attr for input](#), in §4.10.5

[attribute for HTMLInputElement](#), in §4.10.5

[element-attr for button](#), in §4.10.6

[attribute for HTMLButtonElement](#), in §4.10.6

[attribute for HTMLSelectElement](#), in §4.10.7

[attribute for HTMLTextAreaElement](#), in §4.10.11

[attribute for HTMLKeygenElement](#), in §4.10.12

[attribute for HTMLOutputElement](#), in §4.10.13

[attribute for HTMLFieldSetElement](#), in §4.10.16

[element-attr for menu](#), in §4.11.3

[attribute for HTMLMenuElement](#), in §4.11.3

[element-attr for menuitem](#), in §4.11.4

[attribute for HTMLMenuItemElement](#), in §4.11.4

[element-attr for script](#), in §4.12.1

[attribute for HTMLScriptElement](#), in §4.12.1

[attribute for DataTransferItem](#), in §5.7.3.2

[attribute for MIMEType](#), in §7.6.1.5

[element-attr for area](#), in §11.2

[element-attr for param](#), in §11.2

[element-attr for li](#), in §11.2

[element-attr for ul](#), in §11.2

[attribute for HTMLLIElement](#), in §11.3.5

[attribute for HTMLParamElement](#), in §11.3.5

[type information](#), in §2.6.4

[typeMismatch](#), in §4.10.21.3

[typeMustMatch](#), in §4.7.8

[type of the content](#), in §4.7.7

[types](#), in §5.7.3

[type string](#), in §5.7.2

[u](#), in §4.5.24

[ul](#), in §4.4.6

[Unavailable](#), in §4.7.5

[unavailable](#), in §4.7.5

[UNCACHED](#), in §11.3.4.8

[unfocusing steps](#), in §5.4.4

[Unhandled promise rejections](#), in §7.1.3.9

[Unicode character](#), in §2.1.6

[Unicode code point](#), in §2.1.6

[unicode serialization](#), in §6.4

[unicode serialization of an origin](#), in §6.4

[uninitialized](#), in §5.7.3

[unit of related browsing contexts](#), in §6.1.4

[unit of related similar-origin browsing contexts](#), in §6.1.4

[units of related similar-origin browsing contexts](#), in §6.1.4

[unload](#)

[definition of](#), in §6.7.11

[event for global](#), in §Unnumbered section

[unload a document](#), in §6.7.11

[unloaded](#), in §6.7.11

[unloading document cleanup steps](#), in §6.7.11

[unloading document visibility change steps](#), in §6.7.11

[unordered set of unique space-separated tokens](#), in §2.4.7

[unquoted](#), in §8.1.2.3

[unregisterProtocolHandler\(scheme, url\)](#), in §7.6.1.3

[unstyled document](#), in §10.9

[up](#)

[attr-value for marquee/direction](#), in §10.5.11

[state for marquee](#), in §11.3.2

[update\(\)](#), in §11.3.4.8

[update a style block](#), in §4.2.6

[update href](#), in §4.8.3

[UPDATEREADY](#), in §11.3.4.8

[update status](#), in §11.3.4

[update the image data](#), in §4.7.5

[update the session history with the new page](#), in §6.7.1

[update the source set](#), in §4.7.5

[updating the session history with the new page](#), in §6.7.1

[upgrade attempt](#), in §11.3.4.2

[upgrade the pending request to the current request](#), in §4.7.5

[upper-alpha](#), in §4.4.5

[uppercase ASCII hex digits](#), in §2.4.1

[uppercase ASCII letters](#), in §2.4.1

[upper-roman](#), in §4.4.5

[URL](#)

[element-state for input](#), in §4.10.5.1.4

[state for inputmode](#), in §4.10.19.7

[url](#)

[attr-value for input/type](#), in §4.10.5

[value for form/inputmode](#), in §4.10.19.7

[dfn for Location](#), in §6.6.4

[urn](#)

[element-attr for a](#), in §11.2

[element-attr for link](#), in §11.2

[urn:](#), in §2.2.2

[Use Credentials](#), in §2.6.6

[use-credentials](#), in §2.6.6

[used during the parsing](#), in §8.2.5.4.4

[usemap](#)

[element-attr for img](#), in §4.7.5

[element-attr for common](#), in §4.7.17

[element-attr for input](#), in §11.2

[element-attr for object](#), in §11.2

[useMap](#)

[attribute for HTMLImageElement](#), in §4.7.5

[attribute for HTMLInputElement](#), in §11.3.5

[attribute for HTMLObjectElement](#), in §11.3.5

[userAgent](#), in §7.6.1.1

[User agents with no scripting support](#), in §2.2.1

[user interaction task source](#), in §7.1.4.3

[username](#), in §4.8.3

[use srcset or picture](#), in §4.7.5

[using the rules for](#), in §8.2.3.1

[UTF-16BE](#), in §8.2.2.3

[UTF-16 encoding](#), in §2.1.6

[UTF-16LE](#), in §8.2.2.3

[UTF-8](#), in §8.2.2.3

[UTF-8 decode](#), in §2.2.2

[UTF-8 decode without BOM](#), in §2.2.2

[UTF-8 decode without BOM or fail](#), in §2.2.2

[UTF-8 encode](#), in §2.2.2

[valid](#), in §4.10.21.3

[:valid](#), in §4.15.2

[validate the server's response](#), in §2.2.2

[validationMessage](#), in §4.10.21.3

[valid browsing context name](#), in §6.1.5

[valid browsing context name or keyword](#), in §6.1.5

[valid browsing context names or keywords](#), in §6.1.5

[valid date string](#), in §2.4.5.2

[valid date string with optional time](#), in §2.4.5.10

[valid duration string](#), in §2.4.5.9

[valid e-mail address](#), in §4.10.5.1.5

[valid e-mail address list](#), in §4.10.5.1.5

[valid floating date and time string](#), in §2.4.5.5

[valid floating-point number](#), in §2.4.4.3

[valid global date and time string](#), in §2.4.5.7

[valid hash-name reference](#), in §2.4.9

[valid integer](#), in §2.4.4.1

[valid integers](#), in §2.4.4.1

[validity](#), in §4.10.21.3

[ValidityState](#), in §4.10.21.3

[validity states](#), in §4.10.21.1

[valid list of floating-point numbers](#), in §2.4.4.6

[valid lowercase simple color](#), in §2.4.6

[valid media query list](#), in §2.4.10

[valid MIME type](#), in §2.1.1

[valid MIME types with no parameters](#), in §2.1.1

[valid MIME type with no parameters](#), in §2.1.1

[valid month string](#), in §2.4.5.1

[valid non-empty URL](#), in §2.5.1

[valid non-empty URL potentially surrounded by spaces](#), in §2.5.1

[valid non-negative integer](#), in §2.4.4.2

[valid normalized floating date and time string](#), in §2.4.5.5

[valid normalized global date and time string](#), in §2.4.5.7

[valid simple color](#), in §2.4.6

[valid source size list](#), in §4.7.5

[valid time string](#), in §2.4.5.4

[valid time-zone offset string](#), in §2.4.5.6

[valid URL](#), in §2.5.1

[valid URL potentially surrounded by spaces](#), in §2.5.1

[valid week string](#), in §2.4.5.8

[valid yearless date string](#), in §2.4.5.3

vAlign

[attribute for HTMLTableColElement](#), in §11.3.5

[attribute for HTMLTableSectionElement](#), in §11.3.5

[attribute for HTMLTableCellElement](#), in §11.3.5

[attribute for HTMLTableRowElement](#), in §11.3.5

valign

[element-attr for col](#), in §11.2

[element-attr for tbody, thead, tfoot, tablesection](#), in §11.2

[element-attr for td, th, tablecells](#), in §11.2

[element-attr for tr](#), in §11.2

value

[element-attr for li](#), in §4.4.7

[attribute for HTMLLIElement](#), in §4.4.7

[element-attr for data](#), in §4.5.15

[attribute for HTMLDataElement](#), in §4.5.15

[element-attr for param](#), in §4.7.9

[attribute for HTMLParamElement](#), in §4.7.9

[element-attr for input](#), in §4.10.5

[attribute for HTMLInputElement](#), in §4.10.5.4

[mode for input](#), in §4.10.5.4

[element-attr for button](#), in §4.10.6

[attribute for HTMLButtonElement](#), in §4.10.6

[attribute for HTMLSelectElement](#), in §4.10.7

[element-attr for option](#), in §4.10.10

[attribute for HTMLOptionElement](#), in §4.10.10

[attribute for HTMLTextAreaElement](#), in §4.10.11

[mode for output](#), in §4.10.13

[attribute for HTMLOutputElement](#), in §4.10.13

[element-attr for progress](#), in §4.10.14

[attribute for HTMLProgressElement](#), in §4.10.14

[element-attr for meter](#), in §4.10.15

[attribute for HTMLMeterElement](#), in §4.10.15

[dfn for forms](#), in §4.10.18.1

[valueAsDate](#), in §4.10.5.4

[valueAsNumber](#), in §4.10.5.4

[valueMissing](#), in §4.10.21.3

[value mode flag](#), in §4.10.13

[values](#), in §4.10.18.1

[value sanitization algorithm](#), in §4.10.5

[values are reset](#), in §6.7.10

[valuetype](#), in §11.2

[valueType](#), in §11.3.5

[var](#), in §4.5.18

[verbatim](#), in §4.10.19.7

version

[element-attr for html](#), in §11.2

[attribute for HTMLHtmlElement](#), in §11.3.5

[video](#), in §4.7.10

[videoHeight](#), in §4.7.10

[VideoTrack](#), in §4.7.14.10.1

[VideoTrack.id](#), in §4.7.14.10.1

[VideoTrack.kind](#), in §4.7.14.10.1

[VideoTrack.label](#), in §4.7.14.10.1

[VideoTrack.language](#), in §4.7.14.10.1

[VideoTrackList](#), in §4.7.14.10.1

[VideoTrackList.getTrackById\(id\)](#), in §4.7.14.10.1

[VideoTrackList.length](#), in §4.7.14.10.1

[VideoTrackList.selectedIndex](#), in §4.7.14.10.1

[videoTracks](#), in §4.7.14.10

[VideoTrack.selected](#), in §4.7.14.10.1

[videoWidth](#), in §4.7.10

[viewport](#), in §2.2.2

[viewport-based selection](#), in §4.7.1

visible

[definition of](#), in §2.1

[attribute for BarProp](#), in §6.3.6

[:visited](#), in §4.15.2

[Visual user agents that support the suggested default rendering](#), in §2.2.1

[vlink](#), in §11.2

[vLink](#), in §11.3.5

[vlinkColor](#), in §11.3.5

[Void elements](#), in §8.1.2

[volume](#), in §4.7.14.12

[volumechange](#), in §4.7.14.15

vspace

[element-attr for embed](#), in §11.2

[element-attr for iframe](#), in §11.2

[element-attr for input](#), in §11.2

[element-attr for img](#), in §11.2

[element-attr for object](#), in §11.2

[attribute for HTMLAppletElement](#), in §11.3.1

[attribute for HTMLMarqueeElement](#), in §11.3.2

[attribute for HTMLImageElement](#), in §11.3.5

[attribute for HTMLObjectElement](#), in §11.3.5

[waiting](#), in §4.7.14.15

[wbr](#), in §4.5.30

webgl

[context for canvas](#), in §4.12.4

[definition of](#), in §4.12.4

[WebVTT](#), in §2.2.2

[WebVTT file](#), in §2.2.2

[WebVTT file using chapter title text](#), in §2.2.2

[WebVTT file using cue text](#), in §2.2.2

[WebVTT file using only nested cues](#), in §2.2.2

[WebVTT parser](#), in §2.2.2

week

[definition of](#), in §2.4.5.8

[attr-value for input/type](#), in §4.10.5

[Week](#), in §4.10.5.1.9

[week number of the last day](#), in §2.4.5.8

[white_space](#), in §2.4.1

[white_space characters](#), in §2.4.1

width

[attribute for HTMLImageElement](#), in §4.7.5

[attribute for HTMLIFrameElement](#), in §4.7.6

[attribute for HTMLEmbedElement](#), in §4.7.7

[attribute for HTMLObjectElement](#), in §4.7.8

[element-attr for media, img, iframe, embed, object, video, input](#), in §4.7.20

[attribute for media](#), in §4.7.20

[attribute for HTMLInputElement](#), in §4.10.5

[element-attr for canvas](#), in §4.12.4

[attribute for HTMLCanvasElement](#), in §4.12.4

[attribute for ImageBitmap](#), in §7.7

[element-attr for col](#), in §11.2

[element-attr for hr](#), in §11.2

[element-attr for pre](#), in §11.2

[element-attr for table](#), in §11.2

[element-attr for td, th, tablecells](#), in §11.2

[attribute for HTMLAppletElement](#), in §11.3.1

[attribute for HTMLMarqueeElement](#), in §11.3.2

[attribute for HTMLTableColElement](#), in §11.3.5

[attribute for HTMLHRElement](#), in §11.3.5

[attribute for HTMLPreElement](#), in §11.3.5

[attribute for HTMLTableElement](#), in §11.3.5

[attribute for HTMLTableCellElement](#), in §11.3.5

[width descriptor](#), in §4.7.5

[width descriptor value](#), in §4.7.5

[width of the select's labels](#), in §10.5.14

[willful violation](#), in §1.5.2

[willValidate](#), in §4.10.21.3

[window](#), in §6.3

[Window](#), in §6.3

[WindowBase64](#), in §7.2

[WindowEventHandlers](#), in §7.1.5.2.1

[WindowModal](#), in §7.5.3

[WindowProxy](#), in §6.3.7

[windowproxy defineownproperty](#), in §6.3.7.1.6

[windowproxy delete](#), in §6.3.7.1.9

[windowproxy.get](#), in §6.3.7.1.7[windowproxy.getownproperty](#), in §6.3.7.1.5[windowproxy.getprototypeof](#), in §6.3.7.1.1[windowproxy.isextensible](#), in §6.3.7.1.3[windowproxy.ownpropertykeys](#), in §6.3.7.1.10[windowproxy.preventextensions](#), in §6.3.7.1.4[windowproxy.set](#), in §6.3.7.1.8[windowproxy.setprototypeof](#), in §6.3.7.1.2[windows-1250](#), in §8.2.2.3[windows-1251](#), in §8.2.2.3[windows-1252](#), in §8.2.2.3[windows-1254](#), in §8.2.2.3[windows-1256](#), in §8.2.2.3[windows-1257](#), in §8.2.2.3[window.slot](#), in §6.3.7[WindowTimers](#), in §7.4[wrap](#)[element-attr for textarea](#), in §4.10.11[attribute for HTMLTextAreaElement](#), in §4.10.11[wrap callbacks](#), in §7.1.4.2[write\(\)](#), in §7.3.3[writeln\(\)](#), in §7.3.4[XHTML document](#), in §2.1[XHTML documents](#), in §2.1[XLink namespace](#), in §2.8[xml:base](#), in §3.2.5.5[XML-compatible](#), in §2.1.2[XML document](#), in §2.1[XML fragment parsing algorithm](#), in §9.4[XML fragment serialization algorithm](#), in §9.3[XML MIME type](#), in §2.1.2[XML namespace](#), in §2.8[XMLNS namespace](#), in §2.8[XML parser](#), in §9.2[XML scripting support disabled](#), in §9.2[XML scripting support enabled](#), in §9.2[xmp](#), in §11.2[XSLTProcessor](#), in §2.2.2[x-user-defined](#), in §8.2.2.3[yearless date](#), in §2.4.5.3[yet more restrictions](#), in §8.1.2.5

§ Terms defined by reference

[CSP3] defines the following terms:

[content security policy](#)[content security policy directive](#)[content security policy syntax](#)[directives](#)[enforce the policy](#)[enforced](#)[ensurecspdoesnotblockstringcompilation](#)[frame-ancestors](#)[frame-ancestors directive](#)[initialize a document's csp list](#)[initialize a global object's csp list](#)[is base allowed for document?](#)[parse a serialized content security policy](#)[report-uri](#)[sandbox](#)[sandbox directive](#)[should element's inline behavior be blocked by content security policy?](#)[valid content security policy](#)

[CSSOM] defines the following terms:

[alternate flag](#)[alternative style sheet sets](#)[css rules](#)[disabled flag](#)[media](#)[origin-clean flag](#)[owner css rule](#)[owner node](#)[parent css style sheet](#)[preferred style sheet set](#)[serializing a css value](#)[title](#)[type](#)

[CSSOM-VIEW] defines the following terms:

[evaluate media queries and report changes](#)[resize](#)[run the resize steps](#)[run the scroll steps](#)[scroll](#)[the features argument of window.open](#)

[DOM] defines the following terms:

[Element](#)[adopt](#)[append](#)[cd data](#)[clone](#)[cloning](#)[collection](#)[content type](#)[document url](#)[document's character encoding](#)[element attribute](#)[event listener](#)[id](#)[insert](#)[insertion steps](#)[range](#)[range bp](#)[range end](#)[range start](#)[remove](#)[replace](#)[represented by the collection](#)

[DOM-Parsing] defines the following terms:

[DOMParser](#)[innerHTML](#)[outerHTML](#)

[ECMA-262] defines the following terms:

[%arraybuffer%](#)
[%arrayprototype%](#)
[%objproto_tostring%](#)
[%objproto_valueof%](#)
[ArrayBuffer](#)
[Date](#)
[Error](#)
[Function](#)
[RangeError](#)
[RegExp](#)
[SyntaxError](#)
[TypeError](#)
[abstract equality comparison](#)
[arraycreate](#)
[automatic semicolon insertion](#)
[call](#)
[clonearraybuffer](#)
[construct](#)
[createdataproperty](#)
[current realm](#)
[current realm record](#)
[detacharraybuffer](#)
[directive prologue](#)
[early error](#)
[enqueuejob](#)
[functionbody](#)
[functioncreate](#)
[get](#)
[getactivscriptormodule](#)
[getfunctionrealm](#)
[hasownproperty](#)
[hostensurancancompilestrings](#)
[hostpromiserejectiontracker](#)
[initializehostdefinedrealm](#)
[isaccessordescriptor](#)
[iscallable](#)
[isconstructor](#)
[isdatadescriptor](#)
[isdetachedbuffer](#)
[javascript execution context](#)
[javascript execution context stack](#)
[javascript realm](#)
[list](#)
[newobjectenvironment](#)
[ordinarydefineownproperty](#)
[ordinarydelete](#)
[ordinaryget](#)
[ordinarygetownproperty](#)
[ordinarygetprototypeof](#)
[ordinaryisextensible](#)

[ordinaryownpropertykeys](#)
[ordinarypreventextensions](#)
[ordinaryset](#)
[ordinarysetprototypeof](#)
[parsescript](#)
[pattern](#)
[property descriptor](#)
[propertydescriptor](#)
[realm](#)
[record](#)
[runjobs](#)
[running javascript execution context](#)
[samevalue](#)
[script](#)
[scriptevaluation](#)
[strict equality comparison](#)
[the typedarray constructors](#)
[toboolean](#)
[tostring](#)
[toint32](#)
[type](#)
[typedarraycreate](#)
[typeof](#)
[use strict directive](#)
[well-known intrinsic objects](#)
[well-known symbols](#)

[ENCODING] defines the following terms:

[decode](#)
[encode code point](#)
[encoded as utf-8](#)
[encoding](#)

[FETCH] defines the following terms:

[body](#)
[cache mode](#)
[client](#)
[cors protocol](#)
[cors-cross-origin](#)
[credentials mode](#)
[cryptographic nonce metadata](#)
[csp list](#)
[default user-agent value](#)
[destination](#)
[extract a mime type](#)
[fetch](#)
[fetching algorithm](#)
[header list](#)
[https state](#)
[https state value](#)
[initiator](#)
[internal response](#)
[method](#)
[mode](#)
[ok status](#)
[omit-origin-header flag](#)
[origin](#)
[origin header](#)
[parser metadata](#)
[process response](#)
[redirect mode](#)
[referrer](#)
[referrer policy](#)
[request](#)
[requestcredentials](#)
[response](#)
[same-origin data-url flag](#)
[set](#)
[status](#)
[synchronous flag](#)
[target browsing context](#)
[terminate](#)
[type](#)
[unsafe-request flag](#)
[url](#)
[url list](#)
[use-url-credentials flag](#)

[RESOURCE-HINTS] defines the following terms:

[dns-prefetch](#)
[preconnect](#)
[prefetch](#)
[prerender](#)

[SELECTION-API] defines the following terms:

[Selection](#)

[UIEVENTS] defines the following terms:

[FocusEvent](#)

[MouseEvent](#)

[MouseEventInit](#)

[UIEvent](#)

[altKey](#)

[button](#)

[click](#)

[ctrlKey](#)

[dblclick](#)

[detail](#)

[getModifierState\(\)](#)

[keydown](#)

[keypress](#)

[keyup](#)

[metaKey](#)

[mousedown](#)

[mouseenter](#)

[mouseleave](#)

[mousemove](#)

[mouseout](#)

[mouseover](#)

[mouseup](#)

[relatedTarget](#)

[shiftKey](#)

[view](#)

[wheel](#)

[URL] defines the following terms:

[application/x-www-form-urlencoded](#)

[serializer](#)

[basic url parser](#)

[default encode set](#)

[domain](#)

[domain to ascii](#)

[domain to unicode](#)

[fragment](#)

[fragment state](#)

[host](#)

[host equals](#)

[host parser](#)

[host serializer](#)

[host state](#)

[hostname state](#)

[ipv4](#)

[ipv6](#)

[network scheme](#)

[origin](#)

[parse errors](#)

[parsed urls](#)

[path](#)

[path start state](#)

[percent decode](#)

[percent encode](#)

[port](#)

[port state](#)

[query](#)

[query state](#)

[relative schemes](#)

[scheme](#)

[scheme start state](#)

[serialization](#)

[serialize an integer](#)

[serialized](#)

[set the password](#)

[set the username](#)

[url](#)

[url parser](#)

[url serializer](#)

[username](#)

[utf-8 percent encode](#)

[WEBWORKERS] defines the following terms:

[Worker](#)

[WorkerGlobalScope](#)

[closing](#)

[run a worker](#)

[worker event loops](#)

[worker processing model](#)

[XML] defines the following terms:

[name](#)

[xml:space](#)

[WHATWG] defines the following terms:

[whatwg html specification](#)

[aria] defines the following terms:

[alert](#)
[alertdialog](#)
[application](#)
[article](#)
[banner](#)
[button](#)
[checkbox](#)
[columnheader](#)
[combobox](#)
[complementary](#)
[contentinfo](#)
[dialog](#)
[directory](#)
[document](#)
[grid](#)
[gridcell](#)
[group](#)
[heading](#)
[img](#)
[link](#)
[list](#)
[listbox](#)
[listitem](#)
[log](#)
[main](#)
[marquee](#)
[menu](#)
[menubar](#)
[menuitem](#)
[menuitemcheckbox](#)
[menuitemradio](#)
[navigation](#)
[note](#)
[option](#)
[presentation](#)
[progressbar](#)
[radio](#)
[radiogroup](#)
[region](#)
[row](#)
[rowgroup](#)
[rowheader](#)
[search](#)
[separator](#)
[slider](#)
[spinbutton](#)
[status](#)
[tab](#)
[tablist](#)
[tabpanel](#)
[textbox](#)

[toolbar](#)

[tree](#)

[treeitem](#)

[aria-1.1] defines the following terms:

[switch](#)

[css-cascade-3] defines the following terms:

[inherit](#)

[css-color-4] defines the following terms:

[black](#)

[transparent](#)

[CSS3-CONTENT] defines the following terms:

[content](#)

[css-display-3] defines the following terms:

[anonymous](#)

[block](#)

[CSS-FONT-LOADING-3] defines the following terms:

[FontFace](#)

[font source](#)

[CSS-FONTS-3] defines the following terms:

[font-family](#)

[font-size](#)

[css-images-4] defines the following terms:

[object-fit](#)

[css-lists-3] defines the following terms:

[list-style-type](#)

[css-overflow-3] defines the following terms:

[overflow](#)

[CSS3-RUBY] defines the following terms:

[annotation](#)

[ruby annotation](#)

[css-sizing-3] defines the following terms:

[intrinsic sizing](#)

[CSS-STYLE-ATTR] defines the following terms:

[css styling attribute](#)

[CSS-SYNTAX-3] defines the following terms:

[<whitespace-token>](#)

[function](#)

[get an encoding](#)

[parse a comma-separated list of component values](#)

[whitespace](#)

[css-text-3] defines the following terms:

[justify](#)

[pre](#)

[pre-wrap](#)

[text-align](#)

[css-transitions-1] defines the following terms:

[end time](#)

[start time](#)

[css-ui-4] defines the following terms:

[appearance](#)

[CSS-VALUES] defines the following terms:

[<length>](#)

[CSS-WRITING-MODES-3] defines the following terms:

[direction](#)

[ltr](#)

[plaintext](#)

[rtl](#)

[unicode-bidi](#)

[CSS2] defines the following terms:

[auto](#)

[border-spacing](#)

[clear](#)

[color](#)

[css2 system colors](#)

[height](#)

[line-height](#)

[width](#)

[CSS22] defines the following terms:

[background-color](#)
[background-image](#)
[border-bottom-color](#)
[border-bottom-style](#)
[border-bottom-width](#)
[border-left-color](#)
[border-left-style](#)
[border-left-width](#)
[border-right-color](#)
[border-right-style](#)
[border-right-width](#)
[border-top-color](#)
[border-top-style](#)
[border-top-width](#)
[cursor](#)
[display](#)
[letter-spacing](#)
[margin-bottom](#)
[margin-left](#)
[margin-right](#)
[margin-top](#)
[outline](#)
[padding-bottom](#)
[padding-left](#)
[padding-right](#)
[padding-top](#)
[solid](#)
[vertical-align](#)
[white-space](#)

[CSS-SYNTAX-3] defines the following terms:

[component value](#)
[consume a component value](#)
[environment encoding](#)
[whitespace](#)

[CSSOM] defines the following terms:

[CSSStyleDeclaration](#)
[LinkStyle](#)
[StyleSheet](#)
[associated css style sheet](#)
[create a css style sheet](#)
[css style sheet](#)
[cssText](#)
[location](#)
[media](#)
[remove a css style sheet](#)

[CSSOM-VIEW] defines the following terms:

[Screen](#)
[clientY](#)
[screenX](#)
[screenY](#)
[scroll an element into view](#)
[scroll to the beginning of the document](#)

[DOM] defines the following terms:

[Attr](#)
[ChildNode](#)
[Comment](#)
[DOMImplementation](#)
[DOMTokenList](#)
[DocumentFragment](#)
[DocumentType](#)
[Event](#)
[EventInit](#)
[EventTarget](#)
[HTMLCollection](#)
[MutationObserver](#)
[Node](#)
[NodeList](#)
[ProcessingInstruction](#)
[Text](#)
[XMLDocument](#)
[addEventListener\(type, callback\)](#)
[adopting steps](#)
[appendChild\(node\)](#)
[case-sensitively](#)
[childNodes](#)
[classList](#)
[className](#)
[cloneNode\(\)](#)
[cloning steps](#)
[converted to ascii lowercase](#)
[converted to ascii uppercase](#)
[createDocument\(namespace, qualifiedName\)](#)
[createElement\(localName\)](#)
[createElementNS\(namespace, qualifiedName\)](#)
[createHTMLDocument\(\)](#)
[currentTarget](#)
[data](#)
[event](#)
[getElementById\(elementId\)](#)
[getElementsByClassName\(className\)](#)
[id](#)
[importNode\(node, deep\)](#)
[initEvent\(type, bubbles, cancelable\)](#)
[initialize](#)
[insertBefore\(node, child\)](#)
[isTrusted](#)
[item\(index\)](#)
[localName](#)
[namespaceURI](#)
[node document](#)
[other applicable specifications](#)
[ownerDocument](#)

[parentNode](#)
[removing steps](#)
[select](#)
[tagName](#)
[target](#)
[textContent](#)
[traverse](#)
[type](#)
[value](#)

[EVENTSOURCE] defines the following terms:

[EventSource](#)

[FILEAPI] defines the following terms:

[Blob](#)
[File](#)
[FileList](#)
[closed](#)
[type](#)

[FULLSCREEN] defines the following terms:

[fullscreen enabled flag](#)
[fully exit fullscreen](#)
[requestFullscreen\(\)](#)
[top layer](#)

[GEOMETRY-1] defines the following terms:

[DOMMatrix](#)

[HR-TIME-2] defines the following terms:

[DOMHighResTimeStamp](#)
[Performance](#)
[now\(\)](#)

[HTML] defines the following terms:

[BlobCallback](#)
[CanPlayTypeResult](#)
[Document](#)
[DocumentReadyState](#)
[DragEventInit](#)
[ErrorEventInit](#)
[EventHandler](#)
[EventHandlerNonNull](#)
[FrameRequestCallback](#)
[FunctionStringCallback](#)
[HashChangeEventInit](#)
[ImageBitmapSource](#)
[MediaProvider](#)
[PageTransitionEventInit](#)
[PopStateEventInit](#)
[PromiseRejectionEventInit](#)
[RelatedEventInit](#)
[RenderingContext](#)
[ScrollRestoration](#)
[SelectionMode](#)
[TextTrackKind](#)
[TextTrackMode](#)
[TrackEventInit](#)
[activeElement](#)
[addTextTrack\(kind, label\)](#)
[blob](#)
[blur\(\)](#)
[cite](#)
[click\(\)](#)
[colno](#)
[cols](#)
[contentDocument](#)
[contentWindow](#)
[data](#)
[dataTransfer](#)
[dataset](#)
[defaultPlaybackRate](#)
[defaultView](#)
[dir](#)
[ended](#)
[error](#)
[filename](#)
[focus\(\)](#)
[forceSpellCheck\(\)](#)
[height](#)
[hidden](#)
[lang](#)
[lineno](#)
[media](#)
[message](#)
[name](#)

[networkState](#)
[newURL](#)
[noShade](#)
[oldURL](#)
[open\(url, target, features\)](#)
[pause\(\)](#)
[paused](#)
[persisted](#)
[play\(\)](#)
[playbackRate](#)
[played](#)
[poster](#)
[promise](#)
[reason](#)
[relatedTarget](#)
[rows](#)
[sizes](#)
[spellcheck](#)
[src](#)
[srcset](#)
[state](#)
[tabIndex](#)
[title](#)
[toDataURL\(type\)](#)
[track](#)
[translate](#)
[type](#)
[typeMustMatch](#)
[value](#)
[width](#)

[html-1s] defines the following terms:

[MessagePort](#)

[JLREQ] defines the following terms:

[jukugo ruby rendering](#)

[MATHML] defines the following terms:

[annotation-xml](#)
[math](#)
[merror](#)
[mi](#)
[mn](#)
[mo](#)
[ms](#)
[mtext](#)

[MEDIACAPTURE-STREAMS] defines the following terms:

[MediaStream](#)

[mediaqueries-4] defines the following terms:

[<media-condition>](#)

[mediasource] defines the following terms:

[MediaSource](#)

[PAGE-VISIBILITY] defines the following terms:

[hidden](#)

[PROGRESS-EVENTS] defines the following terms:

[ProgressEvent](#)

[promises-guide] defines the following terms:

[resolve](#)

[RFC5988] defines the following terms:

[link header](#)

[COOKIES] defines the following terms:

[cookie header](#)

[receives a set-cookie-string](#)

[receiving a set-cookie-string](#)

[RFC6266] defines the following terms:

[content-disposition](#)

[rfc7230] defines the following terms:

[content-length](#)

[rfc7231] defines the following terms:

[accept](#)

[accept-language](#)

[content-language](#)

[referer](#)

[rfc7232] defines the following terms:

[last-modified](#)

[rfc7234] defines the following terms:

[cache-control](#)

[selectors-4] defines the following terms:

[match](#)

[SVG] defines the following terms:

[SVGMatrix](#)

[SVG2] defines the following terms:

[desc](#)

[foreignobject](#)

[title](#)

[SVGTiny12] defines the following terms:

[process the script element](#)

[TOUCH-EVENTS] defines the following terms:

[Touch](#)

[touch point](#)

[URL] defines the following terms:

[absolute url](#)

[object](#)

[password](#)

[relative url](#)

[url record](#)

[WEBGL] defines the following terms:

[WebGLRenderingContext](#)

[WEBIDL] defines the following terms:

[AbortError](#)

[DOMException](#)

[DOMString](#)

[DataCloneError](#)

[EmptyString](#)

[Exposed](#)

[HierarchyRequestError](#)

[IndexSizeError](#)

[InvalidAccessError](#)

[InvalidCharacterError](#)

[InvalidModificationError](#)

[InvalidNodeTypeError](#)

[InvalidStateError](#)

[LenientThis](#)

[NamespaceError](#)

[NetworkError](#)

[NoInterfaceObject](#)

[NoModificationAllowedError](#)

[NotFoundError](#)

[NotSupportedError](#)

[OverrideBuiltins](#)

[PrimaryGlobal](#)

[Promise](#)

[PutForwards](#)

[QuotaExceededError](#)

[Replaceable](#)

[SameObject](#)

[SecurityError](#)

[TimeoutError](#)

[TreatNonObjectAsNull](#)

[TreatNullAs](#)

[URLMismatchError](#)

[Unforgeable](#)

[WrongDocumentError](#)

[array index property name](#)

[arraybufferview](#)

[callback this value](#)

[converted](#)

[converting](#)

[determine the value of a named property](#)

[determine the value of an indexed property](#)

[global environment associated with](#)

[invoke the web idl callback function](#)

[perform a security check](#)

[platform object](#)

[read only](#)

[support named properties](#)

[supported property indices](#)

[supported property names](#)

[WEBIDL-2] defines the following

terms:

[LegacyUnenumerableNamedPropert](#)
[es](#)
[unenumerable](#)

[XHR] defines the following terms:

[XMLHttpRequest](#)
[fire a progress event named e](#)
[lengthcomputable](#)
[loaded](#)
[responseXML](#)
[total](#)

[XML] defines the following terms:

[document entity](#)
[entity declarations](#)
[entity references](#)
[internal general parsed entity](#)

[XML-STYLESHEET] defines the following terms:

[<?xml-stylesheet?>](#)

§ Elements

This section is non-normative.

List of elements

Element	Description	Categories	Parents†	Children	Attributes	Interface
<a>	Hyperlink	flow ; phrasing* ; interactive	phrasing	transparent*	globals ; href ; target ; download ; rel ; hreflang ; type	HTMLAnchorElement
<abbr>	Abbreviation	flow ; phrasing	phrasing	phrasing	globals	HTMLElement
<address>	Contact information for a page or <article> element	flow	flow	flow*	globals	HTMLElement
<area>	Hyperlink or dead area on an image map	flow ; phrasing	phrasing*	empty	globals ; alt ; coords ; shape ; href ; target ; download ; rel ; hreflang ; type	HTMLAreaElement
<article>	Self-contained syndicable or reusable composition	flow ; sectioning	flow	flow	globals	HTMLElement
<aside>	Sidebar for tangentially related content	flow ; sectioning	flow	flow	globals	HTMLElement
<audio>	Audio player	flow ; phrasing ; embedded ; interactive	phrasing	<source>* ; transparent*	globals ; src ; crossorigin ; preload ; autoplay ; loop ; muted ; controls	HTMLAudioElement
	Keywords	flow ; phrasing	phrasing	phrasing	globals	HTMLElement
<base>	Base URL and default target browsing context for hyperlinks and forms	metadata	<head> ; <template>	empty	globals ; href ; target	HTMLBaseElement
<bdi>	Text directionality isolation	flow ; phrasing	phrasing	phrasing	globals	HTMLElement

Element	Description	Categories	Parents†	Children	Attributes	Interface
<code><bdo></code>	Text directionality formatting	flow ; phrasing	phrasing	phrasing	globals	HTMLElement
<code><blockquote></code>	A section quoted from another source	flow ; sectioning root	flow	flow	globals ; cite	HTMLQuoteElement
<code><body></code>	Document body	sectioning root	<html>	flow	globals ; onafterprint ; onbeforeprint ; onbeforeunload ; onhashchange ; onlanguagechange ; onmessage ; onoffline ; ononline ; onpagehide ; onpageshow ; onpopstate ; onstorage ; onunload	HTMLBodyElement
<code>
</code>	Line break, e.g., in poem or postal address	flow ; phrasing	phrasing	empty	globals	HTMLBRElement
<code><button></code>	Button control	flow ; phrasing ; interactive ; listed ; labelable ; submittable ; reassociateable ; form-associated	phrasing	phrasing *	globals ; autofocus ; disabled ; form ; formation ; formenctype ; formmethod ; formnovalidate ; formtarget ; menu ; name ; type ; value	HTMLButtonElement
<code><canvas></code>	Scriptable bitmap canvas	flow ; phrasing ; embedded	phrasing	transparent	globals ; width ; height	HTMLCanvasElement
<code><caption></code>	Table caption	none	<table> ; <template>	flow *	globals	HTMLTableCaptionElement
<code><cite></code>	Title of a work	flow ; phrasing	phrasing	phrasing	globals	HTMLElement
<code><code></code>	Computer code	flow ; phrasing	phrasing	phrasing	globals	HTMLElement
<code><col></code>	Table column	none	<colgroup> ; <template>	empty	globals ; span	HTMLTableColElement
<code><colgroup></code>	Group of columns in a table	none	<table> ; <template>	<col> *; <template> *	globals ; span	HTMLTableColElement
<code><data></code>	Machine-readable equivalent	flow ; phrasing	phrasing	phrasing	globals ; value	HTMLDataElement
<code><datalist></code>	Container for options for combo box control	flow ; phrasing	phrasing	phrasing ; <option>	globals	HTMLDataListElement

Element	Description	Categories	Parents†	Children	Attributes	Interface
<code><dd></code>	Content for corresponding <code><dt></code> element(s)	none	<code><dl></code> ; <code><template></code>	flow	globals	HTMLInputElement
<code><del></code>	A removal from the document	flow ; phrasing *	phrasing	transparent	globals ; cite ; datetime	HTMLModElement
<code><details></code>	Disclosure control for hiding details	flow ; sectioning root ; interactive	flow	<code><summary></code> *; flow	globals ; open	HTMLDetailsElement
<code><dfn></code>	Defining instance	flow ; phrasing	phrasing	phrasing *	globals	HTMLInputElement
<code><div></code>	Generic flow container	flow	flow	flow	globals	HTMLDivElement
<code><dl></code>	Association list consisting of zero or more name-value groups	flow	flow	<code><dt></code> *; <code><dd></code> *; script-supporting elements	globals	HTMLDListElement
<code><dt></code>	Legend for corresponding <code><dd></code> element(s)	none	<code><dl></code> ; <code><template></code>	flow *	globals	HTMLInputElement
<code></code>	Stress emphasis	flow ; phrasing	phrasing	phrasing	globals	HTMLInputElement
<code><embed></code>	Plugin	flow ; phrasing ; embedded ; interactive	phrasing	empty	globals ; src ; type ; width ; height ; any *	HTMLEmbedElement
<code><fieldset></code>	Group of form controls	flow ; sectioning root ; listed ; reassociateable ; form-associated	flow	<code><legend></code> *; flow	globals ; disabled ; form ; name	HTMLFieldSetElement
<code><figcaption></code>	Caption for <code><figure></code>	none	<code><figure></code> ; <code><template></code>	flow	globals	HTMLInputElement
<code><figure></code>	Figure with optional caption	flow ; sectioning root	flow	<code><figcaption></code> *; flow	globals	HTMLInputElement
<code><footer></code>	Footer for a page or section	flow	flow	flow *	globals	HTMLInputElement
<code><form></code>	User-submittable form	flow	flow	flow *	globals ; accept-charset ; action ; autocomplete ; enctype ; method ; name ; novalidate ; target	HTMLFormElement
<code><h1></code> , <code><h2></code> , <code><h3></code> , <code><h4></code> , <code><h5></code> , <code><h6></code>	Section heading	flow ; heading	flow	phrasing	globals	HTMLHeadingElement

Element	Description	Categories	Parents†	Children	Attributes	Interface
<code><head></code>	Container for document metadata	none	<code><html></code>	<code>metadata</code> <code>content</code> *	globals	HTMLHeadElement
<code><header></code>	Introductory or navigational aids for a page or section	flow	flow	flow *	globals	HTMLHeadingElement
<code><hr></code>	Thematic break	flow	flow	empty	globals	HTMLHRElement
<code><html></code>	Root element	none	none*	<code><head></code> *, <code><body></code> *	globals	HTMLHtmlElement
<code><i></code>	Alternate voice	flow ; phrasing	phrasing	phrasing	globals	HTMLIElement
<code><iframe></code>	Nested browsing context	flow ; phrasing ; embedded ; interactive	phrasing	text*	globals ; src ; srcdoc ; name ; sandbox ; allowfullscreen ; width ; height	HTMLIFrameElement
<code></code>	Image	flow ; phrasing ; embedded ; interactive *; form-associated	phrasing	empty	globals ; alt ; src ; srcset ; crossorigin ; usemap ; ismap ; width ; height	HTMLImageElement
<code><input></code>	Form control	flow ; phrasing ; interactive *; listed ; labelable ; submittable ; resettable ; reassociateable ; form-associated	phrasing	empty	globals ; accept ; alt ; autocomplete ; autofocus ; checked ; dirname ; disabled ; form ; formaction ; formenctype ; formmethod ; formnovalidate ; formtarget ; height ; inputmode ; list ; max ; maxlength ; min ; minlength ; multiple ; name ; pattern ; placeholder ; readonly ; required ; size ; src ; step ; type ; value ; width	HTMLInputElement
<code><ins></code>	An addition to the document	flow ; phrasing *	phrasing	transparent	globals ; cite ; datetime	HTMLModElement
<code><kbd></code>	User input	flow ; phrasing	phrasing	phrasing	globals	HTMLIElement
<code><keygen></code>	Cryptographic key-pair generator form control	flow ; phrasing ; interactive ; listed ; labelable ; submittable ; resettable ; reassociateable	phrasing	empty	globals ; autofocus ; challenge ; disabled ; form ; keytype ; name	HTMLKeygenElement

Element	Description	Categories	Parents†	Children	Attributes	Interface
		form-associated				
<label>	Caption for a form control	flow ; phrasing ; interactive ; reassociateable ; form-associated	phrasing	phrasing *	globals ; for	HTMLLabelElement
<legend>	Caption for <fieldset>	none	<fieldset> ; <template>	phrasing	globals	HTMLLegendElement
	List item	none	 ; ; <menu> *; <template>	flow	globals ; value *	HTMLLIElement
<link>	Link metadata	metadata ; flow *; phrasing *	<head> ; <template> ; <noscript> *; phrasing *	empty	globals ; href ; crossorigin ; rel ; media ; hreflang ; type ; sizes	HTMLLinkElement
<main>	Main content of a document	flow	flow	flow *	globals	HTMLElement
<map>	Image map	flow ; phrasing *	phrasing	transparent ; <area> *	globals ; name	HTMLMapElement
<mark>	Highlight	flow ; phrasing	phrasing	phrasing	globals	HTMLElement
<menu>	Menu of commands	flow	flow ; <menu> *	 *; flow *; <menuitem> *; <hr> *; <menu> *; script-supporting elements *	globals ; type ; label	HTMLMenuElement
<menuitem>	Menu command	none	<menu> ; <template>	empty	globals ; type ; label ; icon ; disabled ; checked ; radiogroup ; default	HTMLMenuItemElement
<meta>	Text metadata	metadata ; flow *; phrasing *	<head> ; <template> ; <noscript> *; phrasing *	empty	globals ; name ; http-equiv ; content ; charset	HTMLMetaElement
<meter>	Gauge	flow ; phrasing ; labelable	phrasing	phrasing *	globals ; value ; min ; max ; low ; high ; optimum	HTMLMeterElement
<nav>	Section with navigational links	flow ; sectioning	flow	flow	globals	HTMLElement
<noscript>	Fallback content for script	metadata ; flow ; phrasing	<head> *; <template> *; phrasing *	varies *	globals	HTMLElement
<object>	Image, nested browsing context , or plugin	flow ; phrasing ; embedded ; interactive *; listed ; submittable ; reassociateable ; form-associated	phrasing	<param> *; transparent	globals ; data ; type ; typemustmatch ; name ; form ; width ; height	HTMLObjectElement

Element	Description	Categories	Parents†	Children	Attributes	Interface
<code></code>	Ordered list	flow	flow	; script-supporting elements	globals ; reversed ; start ; type	HTMLListElement
<code><optgroup></code>	Group of options in a list box	none	<select>; <template>	<option>; script-supporting elements	globals ; disabled ; label	HTMLOptGroupElement
<code><option></code>	Option in a list box or combo box control	none	<select>; <datalist>; <optgroup>; <template>	text*	globals ; disabled ; label ; selected ; value	HTMLOptionElement
<code><output></code>	Calculated output value	flow ; phrasing ; listed ; labelable ; resettable ; reassociateable ; form-associated	phrasing	phrasing	globals ; for ; form ; name	HTMLOutputElement
<code><p></code>	Paragraph	flow	flow	phrasing	globals	HTMLParagraphElement
<code><param></code>	Parameter for <object>	none	<object>; <template>	empty	globals ; name ; value	HTMLParamElement
<code><picture></code>	Image	flow ; phrasing ; embedded	phrasing	<source>* ; one ; script-supporting elements	globals	HTMLPictureElement
<code><pre></code>	Block of preformatted text	flow	flow	phrasing	globals	HTMLPreElement
<code><progress></code>	Progress bar	flow ; phrasing ; labelable	phrasing	phrasing*	globals ; value ; max	HTMLProgressElement
<code><q></code>	Quotation	flow ; phrasing	phrasing	phrasing	globals ; cite	HTMLQuoteElement
<code><rb></code>	Ruby base	none	<ruby>; <template>	phrasing	globals	HTMLElement
<code><rp></code>	Parenthesis for ruby annotation text	none	<ruby>; <rtc>; <template>	phrasing	globals	HTMLElement
<code><rt></code>	Ruby annotation text	none	<ruby>; <rtc>; <template>	phrasing	globals	HTMLElement
<code><rtc></code>	Ruby annotation text container	none	<ruby>; <template>	phrasing	globals	HTMLElement
<code><ruby></code>	Ruby annotation(s)	flow ; phrasing	phrasing	phrasing ; <rp>; <rt>; <rb>; <rtc>*	globals	HTMLElement
<code><s></code>	Inaccurate text	flow ; phrasing	phrasing	phrasing	globals	HTMLElement
<code><samp></code>	Computer output	flow ; phrasing	phrasing	phrasing	globals	HTMLElement

Element	Description	Categories	Parents†	Children	Attributes	Interface
<code><script></code>	Embedded script	metadata ; flow ; phrasing ; script-supporting elements	<head> ; phrasing ; script-supporting elements	script, data, or script documentation*	globals ; src ; type ; charset ; async ; defer ; crossorigin nonce	HTMLScriptElement
<code><section></code>	Generic document or application section	flow ; sectioning	flow	flow	globals	HTMLElement
<code><select></code>	List box control	flow ; phrasing ; interactive ; listed ; labelable ; submittable ; resettable ; reassociateable ; form-associated	phrasing	<option> ; <optgroup> ; script-supporting elements	globals ; autocomplete ; autofocus ; disabled ; form ; multiple ; name ; required ; size	HTMLSelectElement
<code><small></code>	Side comment	flow ; phrasing	phrasing	phrasing	globals	HTMLElement
<code><source></code>	Media source for <video> or <audio>	none	<video> ; <audio> ; <template>	empty	globals ; src ; type	HTMLSourceElement
<code><source></code>	Image source for 	none	<picture>	empty	globals ; srcset ; sizes ; media ; type	HTMLSourceElement
<code></code>	Generic phrasing container	flow ; phrasing	phrasing	phrasing	globals	HTMLSpanElement
<code></code>	Importance	flow ; phrasing	phrasing	phrasing	globals	HTMLElement
<code><style></code>	Embedded styling information	metadata ; flow *	<head> ; <noscript> *; flow *	varies*	globals ; media ; nonce ; type	HTMLStyleElement
<code><sub></code>	Subscript	flow ; phrasing	phrasing	phrasing	globals	HTMLElement
<code><summary></code>	Caption for <details>	none	<details>	phrasing	globals	HTMLElement
<code><sup></code>	Superscript	flow ; phrasing	phrasing	phrasing	globals	HTMLElement
<code><table></code>	Table	flow	flow	<caption> *; <colgroup> *; <thead> *; <tbody> *; <tfoot> *; <tr> *; script-supporting elements	globals ; border	HTMLTableElement
<code><tbody></code>	Group of rows in a table	none	<table> ; <template>	<tr> ; script-supporting elements	globals	HTMLTableSectionElement
<code><td></code>	Table cell	sectioning root	<tr> ; <template>	flow	globals ; colspan ; rowspan ; headers	HTMLTableDataCellElement
<code><template></code>	Template	metadata ; flow ; phrasing ; script-supporting	metadata ; phrasing ; script-supporting	it's complicated*	globals	HTMLTemplateElement

Element	Description	Categories	Parents†	Children	Attributes	Interface
		elements	elements ; <colgroup> *			
<textarea>	Multiline text field	flow ; phrasing ; interactive ; listed ; labelable ; submittable ; resettable ; reassociateable ; form-associated	phrasing	text	globals ; autofocus ; cols ; dirname ; disabled ; form ; inputmode ; maxlength ; minlength ; name ; placeholder ; readonly ; required ; rows ; wrap	HTMLTextAreaElement
<tfoot>	Group of footer rows in a table	none	<table> ; <template>	<tr> ; script-supporting elements	globals	HTMLTableSectionElement
<th>	Table header cell	interactive *	<tr> ; <template>	flow *	globals ; colspan ; rowspan ; headers ; scope ; abbr	HTMLTableHeaderCellElement
<thead>	Group of heading rows in a table	none	<table> ; <template>	<tr> ; script-supporting elements	globals	HTMLTableSectionElement
<time>	Machine-readable equivalent of date- or time-related data	flow ; phrasing	phrasing	phrasing	globals ; datetime	HTMLTimeElement
<title>	Document title	metadata	<head> ; <template>	text *	globals	HTMLTitleElement
<tr>	Table row	none	<table> ; <thead> ; <tbody> ; <tfoot> ; <template>	<th> *; <td> ; script-supporting elements	globals	HTMLTableRowElement
<track>	Timed text track	none	<audio> ; <video> ; <template>	empty	globals ; default ; kind ; label ; src ; srclang	HTMLTrackElement
<u>	Keywords	flow ; phrasing	phrasing	phrasing	globals	HTMLElement
	List	flow	flow	 ; script-supporting elements	globals	HTMLUListElement
<var>	Variable	flow ; phrasing	phrasing	phrasing	globals	HTMLElement
<video>	Video player	flow ; phrasing ; embedded ; interactive	phrasing	<source> *; transparent *	globals ; src ; crossorigin ; poster ; preload ; autoplay ; loop ; muted ; controls ; width ; height	HTMLVideoElement
<wbr>	Line breaking opportunity	flow ; phrasing	phrasing	empty	globals	HTMLElement

An asterisk (*) in a cell indicates that the actual rules are more complicated than indicated in the table above.

† Categories in the "Parents" column refer to parents that list the given categories in their content model, not to elements that themselves are in those categories. For example, the [<a>](#) element's "Parents" column says "phrasing", so any element whose content model contains the "phrasing" category

could be a parent of an [<a>](#) element. Since the "flow" category includes all the "phrasing" elements, that means the [<th>](#) element could be a parent to an [<a>](#) element.

§ Element content categories

This section is non-normative.

List of element content categories

Category	Elements	Elements with exceptions
Metadata content	<base> ; <link> ; <meta> ; <noscript> ; <script> ; <style> ; <template> ; <title>	—
Flow content	<a> ; <abbr> ; <address> ; <article> ; <aside> ; <audio> ; ; <bdi> ; <bdo> ; <blockquote> ;
 ; <button> ; <canvas> ; <cite> ; <code> ; <data> ; <datalist> ; <del> ; <details> ; <dfn> ; <div> ; <dl> ; ; <embed> ; <fieldset> ; <figure> ; <footer> ; <form> ; <h1> ; <h2> ; <h3> ; <h4> ; <h5> ; <h6> ; <header> ; <hr> ; <i> ; <iframe> ; ; <input> ; <ins> ; <kbd> ; <keygen> ; <label> ; <main> ; <map> ; <mark> ; <math> ; <menu> ; <meter> ; <nav> ; <noscript> ; <object> ; ; <output> ; <p> ; <pre> ; <progress> ; <q> ; <ruby> ; <s> ; <samp> ; <script> ; <section> ; <select> ; <small> ; ; ; <sub> ; <sup> ; <svg> ; <table> ; <template> ; <textarea> ; <time> ; <u> ; ; <var> ; <video> ; <wbr> ; Text	<area> (if it is a descendant of a <map> element)
Sectioning content	<article> ; <aside> ; <nav> ; <section>	—
Heading content	<h1> ; <h2> ; <h3> ; <h4> ; <h5> ; <h6> ;	—
Phrasing content	<a> ; <abbr> ; <audio> ; ; <bdi> ; <bdo> ;
 ; <button> ; <canvas> ; <cite> ; <code> ; <data> ; <datalist> ; <del> ; <dfn> ; ; <embed> ; <i> ; <iframe> ; ; <input> ; <ins> ; <kbd> ; <keygen> ; <label> ; <map> ; <mark> ; <math> ; <meter> ; <noscript> ; <object> ; <output> ; <progress> ; <q> ; <ruby> ; <s> ; <samp> ; <script> ; <select> ; <small> ; ; ; <sub> ; <sup> ; <svg> ; <template> ; <textarea> ; <time> ; <u> ; <var> ; <video> ; <wbr> ; Text	<area> (if it is a descendant of a <map> element);
Embedded content	<audio> ; <canvas> ; <embed> ; <iframe> ; ; <math> ; <object> ; <svg> ; <video>	—
Interactive content *	<button> ; <details> ; <embed> ; <iframe> ; <keygen> ; <label> ; <select> ; <textarea>	<a> (if the href attribute is present); <audio> (if the controls attribute is present); (if the usemap attribute is present); <input> (if the type attribute is <i>not</i> in the Hidden state); <video> (if the controls attribute is present)
Sectioning roots	<blockquote> ; <body> ; <details> ; <fieldset> ; <figure> ; <td>	—
Form-associated elements	<button> ; <fieldset> ; <input> ; <keygen> ; <label> ; <object> ; <output> ; <select> ; <textarea> ; 	—
Listed elements	<button> ; <fieldset> ; <input> ; <keygen> ; <object> ; <output> ; <select> ; <textarea>	—
Submittable elements	<button> ; <input> ; <keygen> ; <object> ; <select> ; <textarea>	—
Resettable elements	<input> ; <keygen> ; <output> ; <select> ; <textarea>	—
Labelable elements	<button> ; <input> ; <keygen> ; <meter> ; <output> ; <progress> ; <select> ; <textarea>	—

Category	Elements	Elements with exceptions
Reassociateable elements	<button> ; <fieldset> ; <input> ; <keygen> ; <label> ; <object> ; <output> ; <select> ; <textarea>	—
Palpable content	<a> ; <abbr> ; <address> ; <article> ; <aside> ; ; <bdi> ; <bdo> ; <blockquote> ; <button> ; <canvas> ; <cite> ; <code> ; <data> ; <details> ; <dfn> ; <div> ; ; <embed> ; <fieldset> ; <figure> ; <footer> ; <form> ; <h1> ; <h2> ; <h3> ; <h4> ; <h5> ; <h6> ; <header> ; <i> ; <iframe> ; ; <ins> ; <kbd> ; <keygen> ; <label> ; <main> ; <map> ; <mark> ; <math> ; <meter> ; <nav> ; <object> ; <output> ; <p> ; <pre> ; <progress> ; <q> ; <ruby> ; <s> ; <samp> ; <section> ; <select> ; <small> ; ; ; <sub> ; <sup> ; <svg> ; <table> ; <textarea> ; <time> ; <u> ; <var> ; <video>	<audio> (if the controls attribute is present); <dl> (if the element's children include at least one name-value group); <input> (if the type attribute is <i>not</i> in the Hidden state); (if the element's children include at least one element); (if the element's children include at least one element); Text that is not inter-element whitespace
Script-supporting elements	<script> ; <template>	—

* The [tabindex](#) attribute can also make any element into [interactive content](#).

§ Attributes

This section is non-normative.

List of attributes (excluding event handler content attributes)

Attribute	Element(s)	Description	Value
<abbr>	<th>	Alternative label to use for the header cell when referencing the cell in other contexts	Text *
accept	<input>	Hint for expected file type in file upload controls	Set of comma-separated tokens * consisting of valid MIME types with no parameters or audio/* , video/* , or image/*
accept-charset	<form>	Character encodings to use for form submission	Ordered set of unique space-separated tokens, ASCII case-insensitive, consisting of labels of ASCII-compatible encodings *
accesskey	HTML elements	Keyboard shortcut to activate or focus element	Ordered set of unique space-separated tokens, case-sensitive , consisting of one Unicode code point in length
action	<form>	URL to use for form submission	Valid non-empty URL potentially surrounded by spaces
allowfullscreen	<iframe>	Whether to allow the <iframe> 's contents to use requestFullscreen()	Boolean attribute
alt	<area> ; ; <input>	Replacement text for use when images are not available	Text *
async	<script>	Execute script asynchronously	Boolean attribute
autocomplete	<form>	Default setting for autofill feature for controls in the form	"on" ; "off"
autocomplete	<input> ; <select> ; <textarea>	Hint for form autofill feature	Autofill field name and related tokens*
autofocus	<button> ; <input> ; <keygen> ; <select>	Automatically focus the form control when the page is loaded	Boolean attribute

Attribute	Element(s)	Description	Value
	<textarea>		
autoplay	<audio> ; <video>	Hint that the media resource can be started automatically when the page is loaded	Boolean attribute
border	<table>	Explicit indication that the <table> element is not being used for layout purposes	The empty string, or "1"
challenge	<keygen>	String to package with the generated and signed public key	Text
charset	<meta>	Character encoding declaration	Encoding label *
charset	<script>	Character encoding of the external script resource	Encoding label *
checked	<menuitem> ; <input>	Whether the command or control is checked	Boolean attribute
cite	<blockquote> ; <del> ; <ins> ; <q>	Link to the source of the quotation or more information about the edit	Valid URL potentially surrounded by spaces
class	HTML elements	Classes to which the element belongs	Set of space-separated tokens
cols	<textarea>	Maximum number of characters per line	Valid non-negative integer greater than zero
colspan	<td> ; <th>	Number of columns that the cell is to span	Valid non-negative integer greater than zero
command	<menuitem>	Command definition	ID *
content	<meta>	Value of the element	Text *
contenteditable	HTML elements	Whether the element is editable	"true"; "false"
contextmenu	HTML elements	The element's context menu	ID *
controls	<audio> ; <video>	Show user agent controls	Boolean attribute
coords	<area>	Coordinates for the shape to be created in an image map	Valid list of floating-point numbers *
crossorigin	<audio> ; ; <link> ; <script> ; <video>	How the element handles crossorigin requests	" anonymous "; " use-credentials "
data	<object>	Address of the resource	Valid non-empty URL potentially surrounded by spaces
datetime	<del> ; <ins>	Date and (optionally) time of the change	Valid date string with optional time
datetime	<time>	Machine-readable value	Valid month string , valid date string , valid yearless date string , valid time string , valid floating date and time string , valid time-zone offset string , valid global date and time string , valid week string , valid non-negative integer , or valid duration string
default	<menuitem>	Mark the command as being a default command	Boolean attribute
default	<track>	Enable the track if no other text track is more suitable	Boolean attribute
defer	<script>	Defer script execution	Boolean attribute

Attribute	Element(s)	Description	Value
dir	HTML elements	The text directionality of the element	"ltr"; "rtl"; "auto"
dir	<bdo>	The text directionality of the element	"ltr"; "rtl"
dirname	<input> ; <textarea>	Name of form field to use for sending the element's directionality in form submission	Text*
disabled	<button> ; <menuitem> ; <fieldset> ; <input> ; <keygen> ; <optgroup> ; <option> ; <select> ; <textarea>	Whether the form control is disabled	Boolean attribute
download	<a> ; <area>	Whether to download the resource instead of navigating to it, and its file name if so	Text
draggable	HTML elements	Whether the element is draggable	"true"; "false"
dropzone	HTML elements	Accepted item types for drag-and-drop	Unordered set of unique space-separated tokens, ASCII case-insensitive , consisting of accepted types and drag feedback*
enctype	<form>	Form data set encoding type to use for form submission	"application/x-www-form-urlencoded"; "multipart/form-data"; "text/plain"
for	<label>	Associate the label with form control	ID*
for	<output>	Specifies controls from which the output was calculated	Unordered set of unique space-separated tokens, case-sensitive , consisting of IDs*
form	<button> ; <fieldset> ; <input> ; <keygen> ; <label> ; <object> ; <output> ; <select> ; <textarea>	Associates the control with a <form> element	ID*
formaction	<button> ; <input>	URL to use for form submission	Valid non-empty URL potentially surrounded by spaces
formenctype	<button> ; <input>	Form data set encoding type to use for form submission	"application/x-www-form-urlencoded"; "multipart/form-data"; "text/plain"
formmethod	<button> ; <input>	HTTP method to use for form submission	"GET"; "POST"
formnovalidate	<button> ; <input>	Bypass form control validation for form submission	Boolean attribute
formtarget	<button> ; <input>	Browsing context for form submission	Valid browsing context name or keyword
headers	<td> ; <th>	The header cells for this cell	Unordered set of unique space-separated tokens, case-sensitive , consisting of IDs*
height	<canvas> ; <embed> ; <iframe> ; ; <input> ; <object> ; <video>	Vertical dimension	Valid non-negative integer
hidden	HTML elements	Whether the element is relevant	Boolean attribute
high	<meter>	Low limit of high range	Valid floating-point number *

Attribute	Element(s)	Description	Value
href	<a> ; <area>	Address of the hyperlink	Valid URL potentially surrounded by spaces
href	<link>	Address of the hyperlink	Valid non-empty URL potentially surrounded by spaces
href	<base>	Document base URL	Valid URL potentially surrounded by spaces
hreflang	<a> ; <area> ; <link>	Language of the linked resource	Valid BCP 47 language tag
http-equiv	<meta>	Pragma directive	Text*
icon	<menuitem>	Icon for the command	Valid non-empty URL potentially surrounded by spaces
id	HTML elements	The element's ID	Text*
inputmode	<input> ; <textarea>	Hint for selecting an input modality	"verbatim"; "latin"; "latin-name"; "latin-prose"; "full-width-latin"; "kana"; "kana-name"; "katakana"; "numeric"; "tel"; "email"; "url"
ismap		Whether the image is a server-side image map	Boolean attribute
keytype	<keygen>	The type of cryptographic key to generate	Text*
kind	<track>	The type of text track	"subtitles"; "captions"; "descriptions"; "chapters"; "metadata"
label	<menuitem> ; <menu> ; <optgroup> ; <option> ; <track>	User-visible label	Text
lang	HTML elements	Language of the element	Valid BCP 47 language tag or the empty string
list	<input>	List of autocomplete options	ID*
loop	<audio> ; <video>	Whether to loop the media resource	Boolean attribute
low	<meter>	High limit of low range	Valid floating-point number*
max	<input>	Maximum value	Varies*
max	<meter> ; <progress>	Upper bound of range	Valid floating-point number*
maxlength	<input> ; <textarea>	Maximum length of value	Valid non-negative integer
media	<link> ; <style>	Applicable media	Valid media query list
menu	<button>	Specifies the element's designated pop-up menu	ID*
method	<form>	HTTP method to use for form submission	"get"; "post"
min	<input>	Minimum value	Varies*
min	<meter>	Lower bound of range	Valid floating-point number*
minlength	<input> ; <textarea>	Minimum length of value	Valid non-negative integer
multiple	<input> ; <select>	Whether to allow multiple values	Boolean attribute
muted	<audio> ; <video>	Whether to mute the media resource by default	Boolean attribute
name	<button> ; <fieldset> ; <input> ; <keygen> ; <output> ; <select>	Name of form control to use for form submission and in the form.elements API	Text*

Attribute	Element(s)	Description	Value
	<textarea>		
name	form	Name of form to use in the document. forms API	Text *
name	<iframe> ; <object>	Name of nested browsing context	Valid browsing context name or keyword
name	<map>	Name of image map to reference from the usemap attribute	Text *
name	<meta>	Metadata name	Text *
name	<param>	Name of parameter	Text
nonce	<script> ; <style>	Cryptographic nonce used in Content Security Policy checks [CSP3]	Text
novalidate	<form>	Bypass form control validation for form submission	Boolean attribute
open	<details>	Whether the details are visible	Boolean attribute
optimum	<meter>	Optimum value in gauge	Valid floating-point number *
pattern	<input>	Pattern to be matched by the form control's value	Regular expression matching the JavaScript <i>Pattern</i> production
placeholder	<input> ; <textarea>	User-visible label to be placed within the form control	Text *
poster	<video>	Poster frame to show prior to video playback	Valid non-empty URL potentially surrounded by spaces
preload	<audio> ; <video>	Hints how much buffering the media resource will likely need	"none"; "metadata"; "auto"
radiogroup	<menuitem>	Name of group of commands to treat as a radio button group	Text
readonly	<input> ; <textarea>	Whether to allow the value to be edited by the user	Boolean attribute
rel	<a> ; <area> ; <link>	Relationship of this document (or subsection/topic) to the destination resource	Set of space-separated tokens *
required	<input> ; <select> ; <textarea>	Whether the control is required for form submission	Boolean attribute
rev	<a> ; <link>	Reverse link relationship of the destination resource to this document (or subsection/topic)	Set of space-separated tokens
reversed		Number the list backwards	Boolean attribute
rows	<textarea>	Number of lines to show	Valid non-negative integer greater than zero
rowspan	<td> ; <th>	Number of rows that the cell is to span	Valid non-negative integer
sandbox	<iframe>	Security rules for nested content	Unordered set of unique space-separated tokens, ASCII case-insensitive , consisting of "allow-forms", "allow-pointer-lock", "allow-popups", "allow-same-origin", "allow-scripts and "allow-top-navigation"
spellcheck	HTML elements	Whether the element is to have its spelling and grammar checked	"true"; "false"

Attribute	Element(s)	Description	Value
scope	<th>	Specifies which cells the header cell applies to	"row"; "col"; "rowgroup"; "colgroup"
selected	<option>	Whether the option is selected by default	Boolean attribute
shape	<area>	The kind of shape to be created in an image map	"circle"; "default"; "poly"; "rect"
size	<input> ; <select>	Size of the control	Valid non-negative integer greater than zero
sizes	<link>	Sizes of the icons (for rel="icon")	Unordered set of unique space-separated tokens, ASCII case-insensitive , consisting of sizes*
sizes	 ; <source>	Image sizes for different page layouts	Valid source size list
span	<col> ; <colgroup>	Number of columns spanned by the element	Valid non-negative integer greater than zero
src	<audio> ; <embed> ; <iframe> ; ; <input> ; <script> ; <source> ; <track> ; <video>	Address of the resource	Valid non-empty URL potentially surrounded by spaces
srcdoc	<iframe>	A document to render in the <iframe>	The source of an iframe srcdoc document *
srclang	<track>	Language of the text track	Valid BCP 47 language tag
srcset	 ; <source>	Images to use in different situations (e.g., high-resolution displays, small monitors, etc)	Comma-separated list of image candidate strings
start		Ordinal value of the first item	Valid integer
step	<input>	Granularity to be matched by the form control's value	Valid floating-point number greater than zero, or "any"
style	HTML elements	Presentational and formatting instructions	CSS declarations*
tabindex	HTML elements	Whether the element is focusable, and the relative order of the element for the purposes of sequential focus navigation	Valid integer
target	<a> ; <area>	Browsing context for hyperlink navigation	Valid browsing context name or keyword
target	<base>	Default browsing context for hyperlink navigation and form submission	Valid browsing context name or keyword
target	<form>	Browsing context for form submission	Valid browsing context name or keyword
title	HTML elements	Advisory information for the element	Text
title	<abbr> ; <dfn>	Full term or expansion of abbreviation	Text
title	<input>	Description of pattern (when used with pattern attribute)	Text
title	<menuitem>	Hint describing the command	Text
title	<link>	Title of the link	Text

Attribute	Element(s)	Description	Value
title	<link> ; <style>	Alternative style sheet set name	Text
translate	HTML elements	Whether the element is to be translated when the page is localized	"yes"; "no"
type	<a> ; <area> ; <link>	Hint for the type of the referenced resource	Valid MIME type
type	<button>	Type of button	"submit"; "reset"; "button"; "menu"
type	<embed> ; <object> ; <script> ; <source> ; <style>	Type of embedded resource	Valid MIME type
type	<input>	Type of form control	input type keyword
type	<menu>	Type of menu	"context";
type	<menuitem>	Type of command	"command"; "checkbox"; "radio"
type		Kind of list marker	"1"; "a"; "A"; "i"; "I"
typemustmatch	<object>	Whether the type attribute and the Content-Type value need to match for the resource to be used	Boolean attribute
usemap		Name of image map to use	Valid hash-name reference *
value	<button> ; <option>	Value to be used for form submission	Text
value	<data>	Machine-readable value	Text *
value	<input>	Value of the form control	Varies*
value		Ordinal value of the list item	Valid integer
value	<meter> ; <progress>	Current value of the element	Valid floating-point number
value	<param>	Value of parameter	Text
width	<canvas> ; <embed> ; <iframe> ; ; <input> ; <object> ; <video>	Horizontal dimension	Valid non-negative integer
wrap	<textarea>	How the value of the form control is to be wrapped for form submission	"soft"; "hard"

An asterisk (*) in a cell indicates that the actual rules are more complicated than indicated in the table above.



List of event handler content attributes

Attribute	Element(s)	Description	Value
onabort	HTML elements	abort event handler	Event handler content attribute
onafterprint	<body>	afterprint event handler for Window object	Event handler content attribute
onbeforeprint	<body>	beforeprint event handler for Window object	Event handler content attribute
onbeforeunload	<body>	beforeunload event handler for Window object	Event handler content attribute
onblur	HTML elements	blur event handler	Event handler content attribute

Attribute	Element(s)	Description	Value
oncanplay	HTML elements	canplay event handler	Event handler content attribute
oncanplaythrough	HTML elements	canplaythrough event handler	Event handler content attribute
onchange	HTML elements	change event handler	Event handler content attribute
onclick	HTML elements	click event handler	Event handler content attribute
oncontextmenu	HTML elements	contextmenu event handler	Event handler content attribute
oncopy	HTML elements	copy event handler	Event handler content attribute
oncuechange	HTML elements	cuechange event handler	Event handler content attribute
oncut	HTML elements	cut event handler	Event handler content attribute
ondblclick	HTML elements	dblclick event handler	Event handler content attribute
ondrag	HTML elements	drag event handler	Event handler content attribute
ondragend	HTML elements	dragend event handler	Event handler content attribute
ondragenter	HTML elements	dragenter event handler	Event handler content attribute
ondragexit	HTML elements	dragexit event handler	Event handler content attribute
ondragleave	HTML elements	dragleave event handler	Event handler content attribute
ondragover	HTML elements	dragover event handler	Event handler content attribute
ondragstart	HTML elements	dragstart event handler	Event handler content attribute
ondrop	HTML elements	drop event handler	Event handler content attribute
ondurationchange	HTML elements	durationchange event handler	Event handler content attribute
onended	HTML elements	ended event handler	Event handler content attribute
onerror	HTML elements	error event handler	Event handler content attribute
onfocus	HTML elements	focus event handler	Event handler content attribute
onhashchange	<body>	hashchange event handler for Window object	Event handler content attribute
oninput	HTML elements	input event handler	Event handler content attribute
oninvalid	HTML elements	invalid event handler	Event handler content attribute

Attribute	Element(s)	Description	Value
onkeydown	HTML elements	keydown event handler	Event handler content attribute
onkeypress	HTML elements	keypress event handler	Event handler content attribute
onkeyup	HTML elements	keyup event handler	Event handler content attribute
onlanguagechange	<body>	languagechange event handler for Window object	Event handler content attribute
onload	HTML elements	load event handler	Event handler content attribute
onloadeddata	HTML elements	loadeddata event handler	Event handler content attribute
onloadedmetadata	HTML elements	loadedmetadata event handler	Event handler content attribute
onloadstart	HTML elements	loadstart event handler	Event handler content attribute
onmessage	<body>	message event handler for Window object	Event handler content attribute
onmousedown	HTML elements	mousedown event handler	Event handler content attribute
onmouseenter	HTML elements	mouseenter event handler	Event handler content attribute
onmouseleave	HTML elements	mouseleave event handler	Event handler content attribute
onmousemove	HTML elements	mousemove event handler	Event handler content attribute
onmouseout	HTML elements	mouseout event handler	Event handler content attribute
onmouseover	HTML elements	mouseover event handler	Event handler content attribute
onmouseup	HTML elements	mouseup event handler	Event handler content attribute
onwheel	HTML elements	wheel event handler	Event handler content attribute
onoffline	<body>	offline event handler for Window object	Event handler content attribute
ononline	<body>	online event handler for Window object	Event handler content attribute
onpagehide	<body>	pagehide event handler for Window object	Event handler content attribute
onpageshow	<body>	pageshow event handler for Window object	Event handler content attribute
onpaste	HTML elements	paste event handler	Event handler content attribute
onpause	HTML elements	pause event handler	Event handler content attribute
onplay	HTML elements	play event handler	Event handler content attribute

Attribute	Element(s)	Description	Value
onplaying	HTML elements	playing event handler	Event handler content attribute
onpopstate	<body>	popstate event handler for Window object	Event handler content attribute
onprogress	HTML elements	progress event handler	Event handler content attribute
onratechange	HTML elements	ratechange event handler	Event handler content attribute
onreset	HTML elements	reset event handler	Event handler content attribute
onresize	HTML elements	resize event handler	Event handler content attribute
onscroll	HTML elements	scroll event handler	Event handler content attribute
onseeked	HTML elements	seeked event handler	Event handler content attribute
onseeking	HTML elements	seeking event handler	Event handler content attribute
onselect	HTML elements	select event handler	Event handler content attribute
onshow	HTML elements	show event handler	Event handler content attribute
onstalled	HTML elements	stalled event handler	Event handler content attribute
onstorage	<body>	storage event handler for Window object	Event handler content attribute
onsubmit	HTML elements	submit event handler	Event handler content attribute
onsuspend	HTML elements	suspend event handler	Event handler content attribute
ontimeupdate	HTML elements	timeupdate event handler	Event handler content attribute
ontoggle	HTML elements	toggle event handler	Event handler content attribute
onunload	<body>	unload event handler for Window object	Event handler content attribute
onvolumechange	HTML elements	volumechange event handler	Event handler content attribute
onwaiting	HTML elements	waiting event handler	Event handler content attribute

§ Element Interfaces

This section is non-normative.

List of interfaces for elements

Element(s)	Interface(s)
<a>	HTMLAnchorElement : HTMLElement
<abbr>	HTMLElement

Element(s)	Interface(s)
<code><address></code>	HTMLElement
<code><area></code>	HTMLAreaElement : HTMLElement
<code><article></code>	HTMLElement
<code><aside></code>	HTMLElement
<code><audio></code>	HTMLAudioElement : HTMLMediaElement : HTMLElement
<code></code>	HTMLElement
<code><base></code>	HTMLBaseElement : HTMLElement
<code><bdi></code>	HTMLElement
<code><bdo></code>	HTMLElement
<code><blockquote></code>	HTMLQuoteElement : HTMLElement
<code><body></code>	HTMLBodyElement : HTMLElement
<code>
</code>	HTMLBRElement : HTMLElement
<code><button></code>	HTMLButtonElement : HTMLElement
<code><canvas></code>	HTMLCanvasElement : HTMLElement
<code><caption></code>	HTMLTableCaptionElement : HTMLElement
<code><cite></code>	HTMLElement
<code><code></code>	HTMLElement
<code><col></code>	HTMLTableColElement : HTMLElement
<code><colgroup></code>	HTMLTableColElement : HTMLElement
<code><menuitem></code>	HTMLMenuItemElement : HTMLElement
<code><data></code>	HTMLDataElement : HTMLElement
<code><datalist></code>	HTMLDataListElement : HTMLElement
<code><dd></code>	HTMLElement
<code><del></code>	HTMLModElement : HTMLElement
<code><details></code>	HTMLDetailsElement : HTMLElement
<code><dfn></code>	HTMLElement
<code><div></code>	HTMLDivElement : HTMLElement
<code><dl></code>	HTMLDListElement : HTMLElement
<code><dt></code>	HTMLElement
<code></code>	HTMLElement
<code><embed></code>	HTMLEmbedElement : HTMLElement
<code><fieldset></code>	HTMLFieldSetElement : HTMLElement
<code><figcaption></code>	HTMLElement
<code><figure></code>	HTMLElement
<code><footer></code>	HTMLElement
<code><form></code>	HTMLFormElement : HTMLElement
<code><h1></code>	HTMLHeadingElement : HTMLElement
<code><h2></code>	HTMLHeadingElement : HTMLElement
<code><h3></code>	HTMLHeadingElement : HTMLElement
<code><h4></code>	HTMLHeadingElement : HTMLElement
<code><h5></code>	HTMLHeadingElement : HTMLElement
<code><h6></code>	HTMLHeadingElement : HTMLElement
<code><head></code>	HTMLHeadElement : HTMLElement
<code><header></code>	HTMLElement

Element(s)	Interface(s)
<code><hr></code>	HTMLHRElement : HTMLElement
<code><html></code>	HTMLHtmlElement : HTMLElement
<code><i></code>	HTMLElement
<code><iframe></code>	HTMLIFrameElement : HTMLElement
<code></code>	HTMLImageElement : HTMLElement
<code><input></code>	HTMLInputElement : HTMLElement
<code><ins></code>	HTMLModElement : HTMLElement
<code><kbd></code>	HTMLElement
<code><keygen></code>	HTMLKeygenElement : HTMLElement
<code><label></code>	HTMLLabelElement : HTMLElement
<code><legend></code>	HTMLLegendElement : HTMLElement
<code></code>	HTMLLIElement : HTMLElement
<code><link></code>	HTMLLinkElement : HTMLElement
<code><main></code>	HTMLElement
<code><map></code>	HTMLMapElement : HTMLElement
<code><mark></code>	HTMLElement
<code><menu></code>	HTMLMenuElement : HTMLElement
<code><meta></code>	HTMLMetaElement : HTMLElement
<code><meter></code>	HTMLMeterElement : HTMLElement
<code><nav></code>	HTMLElement
<code><noscript></code>	HTMLElement
<code><object></code>	HTMLObjectElement : HTMLElement
<code></code>	HTMLOListElement : HTMLElement
<code><optgroup></code>	HTMLOptGroupElement : HTMLElement
<code><option></code>	HTMLOptionElement : HTMLElement
<code><output></code>	HTMLOutputElement : HTMLElement
<code><p></code>	HTMLParagraphElement : HTMLElement
<code><param></code>	HTMLParamElement : HTMLElement
<code><picture></code>	HTMLPictureElement : HTMLElement
<code><pre></code>	HTMLPreElement : HTMLElement
<code><progress></code>	HTMLProgressElement : HTMLElement
<code><q></code>	HTMLQuoteElement : HTMLElement
<code><rb></code>	HTMLElement
<code><rp></code>	HTMLElement
<code><rt></code>	HTMLElement
<code><rtc></code>	HTMLElement
<code><ruby></code>	HTMLElement
<code><s></code>	HTMLElement
<code><samp></code>	HTMLElement
<code><script></code>	HTMLScriptElement : HTMLElement
<code><section></code>	HTMLElement
<code><select></code>	HTMLSelectElement : HTMLElement
<code><small></code>	HTMLElement
<code><source></code>	HTMLSourceElement : HTMLElement

Element(s)	Interface(s)
<code><source></code>	HTMLSourceElement : HTMLInputElement
<code></code>	HTMLSpanElement : HTMLInputElement
<code></code>	HTMLInputElement
<code><style></code>	HTMLStyleElement : HTMLInputElement
<code><sub></code>	HTMLInputElement
<code><summary></code>	HTMLInputElement
<code><sup></code>	HTMLInputElement
<code><table></code>	HTMLTableElement : HTMLInputElement
<code><tbody></code>	HTMLTableSectionElement : HTMLInputElement
<code><td></code>	HTMLTableDataCellElement : HTMLTableCellElement : HTMLInputElement
<code><template></code>	HTMLTemplateElement : HTMLInputElement
<code><textarea></code>	HTMLTextAreaElement : HTMLInputElement
<code><tfoot></code>	HTMLTableSectionElement : HTMLInputElement
<code><th></code>	HTMLTableHeaderCellElement : HTMLTableCellElement : HTMLInputElement
<code><thead></code>	HTMLTableSectionElement : HTMLInputElement
<code><time></code>	HTMLTimeElement : HTMLInputElement
<code><title></code>	HTMLTitleElement : HTMLInputElement
<code><tr></code>	HTMLTableRowElement : HTMLInputElement
<code><track></code>	HTMLTrackElement : HTMLInputElement
<code><u></code>	HTMLInputElement
<code></code>	HTMLULListElement : HTMLInputElement
<code><var></code>	HTMLInputElement
<code><video></code>	HTMLVideoElement : HTMLMediaElement : HTMLInputElement
<code><wbr></code>	HTMLInputElement

§ Events

This section is non-normative.

List of events

Event	Interface	Interesting targets	Description
abort	Event	Window	Fired at the Window when the download was aborted by the user
DOMContentLoaded	Event	Document	Fired at the Document once the parser has finished
afterprint	Event	Window	Fired at the Window after printing
afterscriptexecute	Event	<code><script></code> elements	Fired at <code><script></code> elements after the script runs (just before the corresponding load event)
beforeprint	Event	Window	Fired at the Window before printing
beforescriptexecute	Event	<code><script></code> elements	Fired at <code><script></code> elements just before the script runs; canceling the event cancels the

Event	Interface	Interesting targets	Description
			running of the script
beforeunload	BeforeUnloadEvent	Window	Fired at the Window when the page is about to be unloaded, in case the page would like to show a warning prompt
blur	Event	Window , elements	Fired at nodes losing focus
change	Event	Form controls	Fired at controls when the user commits a value change (see also the change event of <input> elements)
click	MouseEvent	Elements	Normally a mouse event; also synthetically fired at an element before its activation behavior is run, when an element is activated from a non-pointer input device (e.g., a keyboard)
contextmenu	Event	Elements	Fired at elements when the user requests their context menu
copy	Event	Elements	Fired at elements when the user copies data to the clipboard
cut	Event	Elements	Fired at elements when the user copies the selected data on the clipboard and removes the selection from the document
error	Event	Global scope objects, Worker objects, elements, networking-related objects	Fired when unexpected errors occur (e.g., networking errors, script errors, decoding errors)
focus	Event	Window , elements	Fired at nodes gaining focus
hashchange	HashChangeEvent	Window	Fired at the Window when the fragment identifier part of the document's address changes
input	Event	Form controls	Fired at controls when the user changes the value (see also the change event of <input> elements)
invalid	Event	Form controls	Fired at controls during form validation if they do not satisfy their constraints
languagechange	Event	Global scope objects	Fired at the global scope object when the user's preferred languages change
load	Event	Window , elements	Fired at the Window when the document has finished loading; fired at an element containing a resource (e.g., , <embed>) when its resource has finished loading
loadend	Event or ProgressEvent	 elements	Fired at elements after a successful load (see also media element events)

Event	Interface	Interesting targets	Description
loadstart	ProgressEvent	 elements	Fired at elements when a load begins (see also media element events)
message	MessageEvent	Window , EventSource, WebSocket, MessagePort, DedicatedWorkerGlobalScope, Worker	Fired at an object when it receives a message
offline	Event	Global scope objects	Fired at the global scope object when the network connections fails
online	Event	Global scope objects	Fired at the global scope object when the network connections returns
open	Event	EventSource, WebSocket	Fired at networking-related objects when a connection is established
pagehide	PageTransitionEvent	Window	Fired at the Window when the page's entry in the session history stops being the current entry
pageshow	PageTransitionEvent	Window	Fired at the Window when the page's entry in the session history becomes the current entry
paste	Event	Elements	Fired at elements when the user will insert the clipboard data in the most suitable format (if any) supported for the given context
popstate	PopStateEvent	Window	Fired at the Window when the user navigates the session history
progress	ProgressEvent	 elements	Fired at elements during a CORS-same-origin image load (see also media element events)
readystatechange	Event	Document	Fired at the Document when it finishes parsing and again when all its subresources have finished loading
reset	Event	<form> elements	Fired at a <form> element when it is reset
select	Event	Form controls	Fired at form controls when their text selection is adjusted (whether by an API or by the user)
show	RelatedEvent	<menu> elements	Fired at a <menu> element when it is shown as a context menu
storage	StorageEvent	Window	Fired at Window event when the corresponding <code>localStorage</code> or <code>sessionStorage</code> storage areas

Event	Interface	Interesting targets	Description
			change
submit	Event	<form> elements	Fired at a <form> element when it is submitted
toggle	Event	<details> element	Fired at <details> elements when they open or close
unload	Event	Window	Fired at the Window object when the page is going away

See also [media element events](#), [application cache events](#), and [drag-and-drop events](#).

§ IDL Index

```

[LegacyUnenumerableNamedProperties]
interface HTMLAllCollection {
  readonly attribute unsigned long length;
  getter Element? (unsigned long index);
  getter (HTMLCollection or Element)? namedItem(DOMString name);
  legacycaller (HTMLCollection or Element)? item(optional DOMString nameOrItem);
};

interface HTMLFormControlsCollection : HTMLCollection {
  // inherits length and item()
  getter (RadioNodeList or Element)? namedItem(DOMString name); // shadows inherited namedItem()
};

interface RadioNodeList : NodeList {
  attribute DOMString value;
};

interface HTMLOptionsCollection : HTMLCollection {
  // inherits item(), namedItem()
  attribute unsigned long length; // shadows inherited length
  setter void (unsigned long index, HTMLOptionElement? option);
  void add((HTMLOptionElement or HTMLOptGroupElement) element, optional (HTMLElement or long)?
before = null);
  void remove(long index);
  attribute long selectedIndex;
};

[OverrideBuiltins]
interface DOMStringMap {
  getter DOMString (DOMString name);
  setter void (DOMString name, DOMString value);
  deleter void (DOMString name);
};

interface DOMElementMap {
  getter Element (DOMString name);
  setter creator void (DOMString name, Element value);
  deleter void (DOMString name);
};

enum DocumentReadyState { "loading", "interactive", "complete" };

[OverrideBuiltins]
partial /*sealed*/ interface Document {
  // resource metadata management
  [PutForwards=href, Unforgeable] readonly attribute Location? location;
  attribute DOMString domain;
  readonly attribute DOMString referrer;
  attribute DOMString cookie;
  readonly attribute DOMString lastModified;
  readonly attribute DocumentReadyState readyState;

  // DOM tree accessors
  getter object (DOMString name);
  attribute DOMString title;
  attribute DOMString dir;
  attribute HTMLElement? body;
  readonly attribute HTMLHeadElement? head;
  [SameObject] readonly attribute HTMLCollection images;
  [SameObject] readonly attribute HTMLCollection embeds;
  [SameObject] readonly attribute HTMLCollection plugins;
  [SameObject] readonly attribute HTMLCollection links;
  [SameObject] readonly attribute HTMLCollection forms;
  [SameObject] readonly attribute HTMLCollection scripts;

```



```

    NodeList getElementsByName(DOMString elementName);
    readonly attribute HTMLScriptElement? currentScript;

    // dynamic markup insertion
    Document open(optional DOMString type = "text/html", optional DOMString replace = "");
    WindowProxy open(DOMString url, DOMString name, DOMString features, optional boolean replace =
false);
    void close();
    void write(DOMString... text);
    void writeln(DOMString... text);

    // user interaction
    readonly attribute WindowProxy? defaultView;
    readonly attribute Element? activeElement;
    boolean hasFocus();
    attribute DOMString designMode;
    boolean execCommand(DOMString commandId, optional boolean showUI = false, optional DOMString
value = "");
    boolean queryCommandEnabled(DOMString commandId);
    boolean queryCommandIndeterm(DOMString commandId);
    boolean queryCommandState(DOMString commandId);
    boolean queryCommandSupported(DOMString commandId);
    DOMString queryCommandValue(DOMString commandId);

    // special event handler IDL attributes that only apply to Document objects
    [LenientThis] attribute EventHandler onreadystatechange;
};
Document implements GlobalEventHandlers;
Document implements DocumentAndElementEventHandlers;

partial interface XMLDocument {
    boolean load(DOMString url);
};

interface HTMLElement : Element {
    // metadata attributes
    attribute DOMString title;
    attribute DOMString lang;
    attribute boolean translate;
    attribute DOMString dir;
    [SameObject] readonly attribute DOMStringMap dataset;

    // user interaction
    attribute boolean hidden;
    void click();
    attribute long tabIndex;
    void focus();
    void blur();
    attribute DOMString accessKey;
    attribute boolean draggable;
    [PutForwards=value] readonly attribute DOMTokenList dropzone;
    attribute HTMLMenuElement? contextMenu;
    attribute boolean spellcheck;
    void forceSpellCheck();
};
HTMLElement implements GlobalEventHandlers;
HTMLElement implements DocumentAndElementEventHandlers;
HTMLElement implements ElementContentEditable;

interface HTMLUnknownElement : HTMLElement { };

interface HTMLHtmlElement : HTMLElement { };

interface HTMLHeadElement : HTMLElement { };

```

```
interface HTMLTitleElement : HTMLElement {
    attribute DOMString text;
};

interface HTMLBaseElement : HTMLElement {
    attribute DOMString href;
    attribute DOMString target;
};

interface HTMLLinkElement : HTMLElement {
    attribute DOMString href;
    attribute DOMString? crossOrigin;
    attribute DOMString rel;
    attribute DOMString rev;
    [SameObject, PutForwards=value]readonly attribute DOMTokenList relList;
    attribute DOMString media;
    attribute DOMString hreflang;
    attribute DOMString type;
    [SameObject, PutForwards=value] readonly attribute DOMTokenList sizes;
};
HTMLLinkElement implements LinkStyle;

interface HTMLMetaElement : HTMLElement {
    attribute DOMString name;
    attribute DOMString httpEquiv;
    attribute DOMString content;
};

interface HTMLStyleElement : HTMLElement {
    attribute DOMString media;
    attribute DOMString nonce;
    attribute DOMString type;
};
HTMLStyleElement implements LinkStyle;

interface HTMLBodyElement : HTMLElement {
};
HTMLBodyElement implements WindowEventHandlers;

interface HTMLHeadingElement : HTMLElement {};

interface HTMLParagraphElement : HTMLElement {};

interface HTMLHRElement : HTMLElement {};

interface HTMLPreElement : HTMLElement {};

interface HTMLQuoteElement : HTMLElement {
    attribute DOMString cite;
};

interface HTMLLOListElement : HTMLElement {
    attribute boolean reversed;
    attribute long start;
    attribute DOMString type;
};

interface HTMLULListElement : HTMLElement {};

interface HTMLLIElement : HTMLElement {
    attribute long value;
};
```

```
interface HTMLDListElement : HTMLElement {}

interface HTMLDivElement : HTMLElement {}

interface HTMLAnchorElement : HTMLElement {
    attribute DOMString target;
    attribute DOMString download;
    attribute DOMString rel;
    attribute DOMString rev;
    [SameObject, PutForwards=value] readonly attribute DOMTokenList relList;
    attribute DOMString hreflang;
    attribute DOMString type;
    attribute DOMString text;
};
HTMLAnchorElement implements HTMLHyperlinkElementUtils;

interface HTMLDataElement : HTMLElement {
    attribute DOMString value;
};

interface HTMLTimeElement : HTMLElement {
    attribute DOMString dateTime;
};

interface HTMLSpanElement : HTMLElement {};

interface HTMLBRElement : HTMLElement {};

interface HTMLModElement : HTMLElement {
    attribute DOMString cite;
    attribute DOMString dateTime;
};

interface HTMLPictureElement : HTMLElement {};

partial interface HTMLSourceElement {
    attribute DOMString srcset;
    attribute DOMString sizes;
    attribute DOMString media;
};

[NamedConstructor=Image(optional unsigned long width, optional unsigned long height)]
interface HTMLImageElement : HTMLElement {
    attribute DOMString alt;
    attribute DOMString src;
    attribute DOMString srcset;
    attribute DOMString sizes;
    attribute DOMString? crossOrigin;
    attribute DOMString useMap;
    attribute boolean isMap;
    attribute unsigned long width;
    attribute unsigned long height;
    readonly attribute unsigned long naturalWidth;
    readonly attribute unsigned long naturalHeight;
    readonly attribute boolean complete;
    readonly attribute DOMString currentSrc;
};

interface HTMLIFrameElement : HTMLElement {
    attribute DOMString src;
    attribute DOMString srcdoc;
    attribute DOMString name;
    [PutForwards=value] readonly attribute DOMTokenList sandbox;
    attribute boolean allowFullscreen;
```

```
    attribute DOMString width;
    attribute DOMString height;
    readonly attribute Document? contentDocument;
    readonly attribute WindowProxy? contentWindow;
};

interface HTMLEmbedElement : HTMLElement {
    attribute DOMString src;
    attribute DOMString type;
    attribute DOMString width;
    attribute DOMString height;
    legacycaller any (any... arguments);
};

interface HTMLObjectElement : HTMLElement {
    attribute DOMString data;
    attribute DOMString type;
    attribute boolean typeMustMatch;
    attribute DOMString name;
    readonly attribute HTMLFormElement? form;
    attribute DOMString width;
    attribute DOMString height;
    readonly attribute Document? contentDocument;
    readonly attribute WindowProxy? contentWindow;

    readonly attribute boolean willValidate;
    readonly attribute ValidityState validity;
    readonly attribute DOMString validationMessage;
    boolean checkValidity();
    boolean reportValidity();
    void setCustomValidity(DOMString error);

    legacycaller any (any... arguments);
};

interface HTMLParamElement : HTMLElement {
    attribute DOMString name;
    attribute DOMString value;
};

interface HTMLVideoElement : HTMLMediaElement {
    attribute unsigned long width;
    attribute unsigned long height;
    readonly attribute unsigned long videoWidth;
    readonly attribute unsigned long videoHeight;
    attribute DOMString poster;
};

[NamedConstructor=Audio(optional DOMString src)]
interface HTMIAudioElement : HTMLMediaElement {};

interface HTMLSourceElement : HTMLElement {
    attribute DOMString src;
    attribute DOMString type;
};

interface HTMLTrackElement : HTMLElement {
    attribute DOMString kind;
    attribute DOMString src;
    attribute DOMString srclang;
    attribute DOMString label;
    attribute boolean default;

    const unsigned short NONE = 0;
```

```
const unsigned short LOADING = 1;
const unsigned short LOADED = 2;
const unsigned short ERROR = 3;
readonly attribute unsigned short readyState;

    readonly attribute TextTrack track;
};

enum CanPlayTypeResult { "" /* empty string */, "maybe", "probably" };

typedef (MediaStream or MediaSource or Blob) MediaProvider;

interface HTMLMediaElement : HTMLElement {

    // error state
    readonly attribute MediaError? error;

    // network state
    attribute DOMString src;
    attribute MediaProvider? srcObject;
    readonly attribute DOMString currentSrc;
    attribute DOMString? crossOrigin;
    const unsigned short NETWORK_EMPTY = 0;
    const unsigned short NETWORK_IDLE = 1;
    const unsigned short NETWORK_LOADING = 2;
    const unsigned short NETWORK_NO_SOURCE = 3;
    readonly attribute unsigned short networkState;
    attribute DOMString preload;
    readonly attribute TimeRanges buffered;
    void load();
    CanPlayTypeResult canPlayType(DOMString type);

    // ready state
    const unsigned short HAVE_NOTHING = 0;
    const unsigned short HAVE_METADATA = 1;
    const unsigned short HAVE_CURRENT_DATA = 2;
    const unsigned short HAVE_FUTURE_DATA = 3;
    const unsigned short HAVE_ENOUGH_DATA = 4;
    readonly attribute unsigned short readyState;
    readonly attribute boolean seeking;

    // playback state
    attribute double currentTime;
    void fastSeek(double time);
    readonly attribute unrestricted double duration;
    object getStartDate();
    readonly attribute boolean paused;
    attribute double defaultPlaybackRate;
    attribute double playbackRate;
    readonly attribute TimeRanges played;
    readonly attribute TimeRanges seekable;
    readonly attribute boolean ended;
    attribute boolean autoplay;
    attribute boolean loop;
    void play();
    void pause();

    // controls
    attribute boolean controls;
    attribute double volume;
    attribute boolean muted;
    attribute boolean defaultMuted;

    // tracks
```

```
[SameObject] readonly attribute AudioTrackList audioTracks;
[SameObject] readonly attribute VideoTrackList videoTracks;
[SameObject] readonly attribute TextTrackList textTracks;
TextTrack addTextTrack(TextTrackKind kind, optional DOMString label = "", optional DOMString
language = "");
};

interface MediaError {
    const unsigned short MEDIA_ERR_ABORTED = 1;
    const unsigned short MEDIA_ERR_NETWORK = 2;
    const unsigned short MEDIA_ERR_DECODE = 3;
    const unsigned short MEDIA_ERR_SRC_NOT_SUPPORTED = 4;
    readonly attribute unsigned short code;
};

interface AudioTrackList : EventTarget {
    readonly attribute unsigned long length;
    getter AudioTrack (unsigned long index);
    AudioTrack? getTrackById(DOMString id);

    attribute EventHandler onchange;
    attribute EventHandler onaddtrack;
    attribute EventHandler onremovetrack;
};

interface AudioTrack {
    readonly attribute DOMString id;
    readonly attribute DOMString kind;
    readonly attribute DOMString label;
    readonly attribute DOMString language;
    attribute boolean enabled;
};

interface VideoTrackList : EventTarget {
    readonly attribute unsigned long length;
    getter VideoTrack (unsigned long index);
    VideoTrack? getTrackById(DOMString id);
    readonly attribute long selectedIndex;

    attribute EventHandler onchange;
    attribute EventHandler onaddtrack;
    attribute EventHandler onremovetrack;
};

interface VideoTrack {
    readonly attribute DOMString id;
    readonly attribute DOMString kind;
    readonly attribute DOMString label;
    readonly attribute DOMString language;
    attribute boolean selected;
};

interface TextTrackList : EventTarget {
    readonly attribute unsigned long length;
    getter TextTrack (unsigned long index);
    TextTrack? getTrackById(DOMString id);

    attribute EventHandler onchange;
    attribute EventHandler onaddtrack;
    attribute EventHandler onremovetrack;
};

enum TextTrackMode { "disabled", "hidden", "showing" };
```

```

enum TextTrackKind { "subtitles", "captions", "descriptions", "chapters", "metadata" };

interface TextTrack : EventTarget {
    readonly attribute TextTrackKind kind;
    readonly attribute DOMString label;
    readonly attribute DOMString language;

    readonly attribute DOMString id;
    readonly attribute DOMString inBandMetadataTrackDispatchType;

    attribute TextTrackMode mode;

    readonly attribute TextTrackCueList? cues;
    readonly attribute TextTrackCueList? activeCues;

    void addCue(TextTrackCue cue);
    void removeCue(TextTrackCue cue);

    attribute EventHandler oncuechange;
};

interface TextTrackCueList {
    readonly attribute unsigned long length;
    getter TextTrackCue (unsigned long index);
    TextTrackCue? getCueById(DOMString id);
};

interface TextTrackCue : EventTarget {
    readonly attribute TextTrack? track;

    attribute DOMString id;
    attribute double startTime;
    attribute double endTime;
    attribute boolean pauseOnExit;

    attribute EventHandler onenter;
    attribute EventHandler onexit;
};

[Constructor(double startTime, double endTime, ArrayBuffer data)]
interface DataCue : TextTrackCue {
    attribute ArrayBuffer data;
};

interface TimeRanges {
    readonly attribute unsigned long length;
    double start(unsigned long index);
    double end(unsigned long index);
};

[Constructor(DOMString type, optional TrackEventInit eventInitDict)]
interface TrackEvent : Event {
    readonly attribute (VideoTrack or AudioTrack or TextTrack)? track;
};

dictionary TrackEventInit : EventInit {
    (VideoTrack or AudioTrack or TextTrack)? track;
};

interface HTMLMapElement : HTMLElement {
    attribute DOMString name;
    [SameObject] readonly attribute HTMLCollection areas;
    [SameObject] readonly attribute HTMLCollection images;
};

```

```
interface HTMLAreaElement : HTMLElement {
    attribute DOMString alt;
    attribute DOMString coords;
    attribute DOMString shape;
    attribute DOMString target;
    attribute DOMString download;
    attribute DOMString rel;
    readonly attribute DOMTokenList relList;
    attribute DOMString hreflang;
    attribute DOMString type;
};
HTMLAreaElement implements HTMLHyperlinkElementUtils;

[NoInterfaceObject]
interface HTMLHyperlinkElementUtils {
    stringifier attribute USVString href;
    readonly attribute USVString origin;
    attribute USVString protocol;
    attribute USVString username;
    attribute USVString password;
    attribute USVString host;
    attribute USVString hostname;
    attribute USVString port;
    attribute USVString pathname;
    attribute USVString search;
    attribute USVString hash;
};

interface HTMLTableElement : HTMLElement {
    attribute HTMLTableCaptionElement? caption;
    HTMLTableCaptionElement createCaption();
    void deleteCaption();
    attribute HTMLTableSectionElement? tHead;
    HTMLTableSectionElement createTHead();
    void deleteTHead();
    attribute HTMLTableSectionElement? tFoot;
    HTMLTableSectionElement createTFoot();
    void deleteTFoot();
    [SameObject] readonly attribute HTMLCollection tBodies;
    HTMLTableSectionElement createTBody();
    [SameObject] readonly attribute HTMLCollection rows;
    HTMLTableRowElement insertRow(optional long index = -1);
    void deleteRow(long index);
};

interface HTMLTableCaptionElement : HTMLElement {};

interface HTMLTableColElement : HTMLElement {
    attribute unsigned long span;
};

interface HTMLTableSectionElement : HTMLElement {
    [SameObject] readonly attribute HTMLCollection rows;
    HTMLElement insertRow(optional long index = -1);
    void deleteRow(long index);
};

interface HTMLTableRowElement : HTMLElement {
    readonly attribute long rowIndex;
    readonly attribute long sectionRowIndex;
    [SameObject] readonly attribute HTMLCollection cells;
    HTMLElement insertCell(optional long index = -1);
    void deleteCell(long index);
};
```



```
};

interface HTMLTableDataCellElement : HTMLTableCellElement {};

interface HTMLTableHeaderCellElement : HTMLTableCellElement {
    attribute DOMString scope;
    attribute DOMString abbr;
};

interface HTMLTableCellElement : HTMLElement {
    attribute unsigned long colSpan;
    attribute unsigned long rowSpan;
    [PutForwards=value] readonly attribute DOMTokenList headers;
    readonly attribute long cellIndex;
};

[OverrideBuiltins]
interface HTMLFormElement : HTMLElement {
    attribute DOMString acceptCharset;
    attribute DOMString action;
    attribute DOMString autoComplete;
    attribute DOMString enctype;
    attribute DOMString encoding;
    attribute DOMString method;
    attribute DOMString name;
    attribute boolean noValidate;
    attribute DOMString target;

    [SameObject] readonly attribute HTMLFormControlsCollection elements;
    readonly attribute unsigned long length;
    getter Element (unsigned long index);
    getter (RadioNodeList or Element) (DOMString name);

    void submit();
    void reset();
    boolean checkValidity();
    boolean reportValidity();
};

interface HTMLLabelElement : HTMLElement {
    readonly attribute HTMLFormElement? form;
    attribute DOMString htmlFor;
    readonly attribute HTMLElement? control;
};

interface HTMLInputElement : HTMLElement {
    attribute DOMString accept;
    attribute DOMString alt;
    attribute DOMString autoComplete;
    attribute boolean autofocus;
    attribute boolean defaultChecked;
    attribute boolean checked;
    attribute DOMString dirName;
    attribute boolean disabled;
    readonly attribute HTMLFormElement? form;
    readonly attribute FileList? files;
    attribute DOMString formAction;
    attribute DOMString formEnctype;
    attribute DOMString formMethod;
    attribute boolean formNoValidate;
    attribute DOMString formTarget;
    attribute unsigned long height;
    attribute boolean indeterminate;
    attribute DOMString inputMode;
```

```

readonly attribute HTMLElement? list;
attribute DOMString max;
attribute long maxLength;
attribute DOMString min;
attribute long minLength;
attribute boolean multiple;
attribute DOMString name;
attribute DOMString pattern;
attribute DOMString placeholder;
attribute boolean readOnly;
attribute boolean required;
attribute unsigned long size;
attribute DOMString src;
attribute DOMString step;
attribute DOMString type;
attribute DOMString defaultValue;
[TreatNullAs=EmptyString] attribute DOMString value;
attribute object? valueAsDate;
attribute unrestricted double valueAsNumber;
attribute unsigned long width;

void stepUp(optional long n = 1);
void stepDown(optional long n = 1);

readonly attribute boolean willValidate;
readonly attribute ValidityState validity;
readonly attribute DOMString validationMessage;
boolean checkValidity();
boolean reportValidity();
void setCustomValidity(DOMString error);

[SameObject] readonly attribute NodeList labels;

void select();
attribute unsigned long selectionStart;
attribute unsigned long selectionEnd;
attribute DOMString selectionDirection;
void setRangeText(DOMString replacement);
void setRangeText(DOMString replacement, unsigned long start, unsigned long end, optional
SelectionMode selectionMode = "preserve");
void setSelectionRange(unsigned long start, unsigned long end, optional DOMString direction);
};

interface HTMLButtonElement : HTMLElement {
  attribute boolean autofocus;
  attribute boolean disabled;
  readonly attribute HTMLFormElement? form;
  attribute DOMString formAction;
  attribute DOMString formEnctype;
  attribute DOMString formMethod;
  attribute boolean formNoValidate;
  attribute DOMString formTarget;
  attribute DOMString name;
  attribute DOMString type;
  attribute DOMString value;
  attribute HTMLMenuElement? menu;

  readonly attribute boolean willValidate;
  readonly attribute ValidityState validity;
  readonly attribute DOMString validationMessage;
  boolean checkValidity();
  boolean reportValidity();
  void setCustomValidity(DOMString error);

```

```

    [SameObject] readonly attribute NodeList labels;
};

interface HTMLSelectElement : HTMLElement {
    attribute DOMString autocomplete;
    attribute boolean autofocus;
    attribute boolean disabled;
    readonly attribute HTMLFormElement? form;
    attribute boolean multiple;
    attribute DOMString name;
    attribute boolean required;
    attribute unsigned long size;

    readonly attribute DOMString type;

    [SameObject] readonly attribute HTMLOptionsCollection options;
    attribute unsigned long length;
    getter Element? item(unsigned long index);
    HTMLOptionElement? namedItem(DOMString name);
    void add((HTMLOptionElement or HTMLOptGroupElement) element, optional (HTMLElement or long)?
before = null);
    void remove(); // ChildNode overload
    void remove(long index);
    setter void (unsigned long index, HTMLOptionElement? option);

    [SameObject] readonly attribute HTMLCollection selectedOptions;
    attribute long selectedIndex;
    attribute DOMString value;

    readonly attribute boolean willValidate;
    readonly attribute ValidityState validity;
    readonly attribute DOMString validationMessage;
    boolean checkValidity();
    boolean reportValidity();
    void setCustomValidity(DOMString error);

    [SameObject] readonly attribute NodeList labels;
};

interface HTMLDataListElement : HTMLElement {
    [SameObject] readonly attribute HTMLCollection options;
};

interface HTMLOptGroupElement : HTMLElement {
    attribute boolean disabled;
    attribute DOMString label;
};

[NamedConstructor=Option(optional DOMString text = "", optional DOMString value, optional
boolean defaultSelected = false, optional boolean selected = false)]
interface HTMLOptionElement : HTMLElement {
    attribute boolean disabled;
    readonly attribute HTMLFormElement? form;
    attribute DOMString label;
    attribute boolean defaultSelected;
    attribute boolean selected;
    attribute DOMString value;

    attribute DOMString text;
    readonly attribute long index;
};

interface HTMLTextAreaElement : HTMLElement {
    attribute DOMString autocomplete;

```

```

    attribute boolean autofocus;
    attribute unsigned long cols;
    attribute DOMString dirName;
    attribute boolean disabled;
    readonly attribute HTMLFormElement? form;
    attribute DOMString inputMode;
    attribute long maxLength;
    attribute long minLength;
    attribute DOMString name;
    attribute DOMString placeholder;
    attribute boolean readOnly;
    attribute boolean required;
    attribute unsigned long rows;
    attribute DOMString wrap;

    readonly attribute DOMString type;
    attribute DOMString defaultValue;
    [TreatNullAs=EmptyString] attribute DOMString value;
    readonly attribute unsigned long textLength;

    readonly attribute boolean willValidate;
    readonly attribute ValidityState validity;
    readonly attribute DOMString validationMessage;
    boolean checkValidity();
    boolean reportValidity();
    void setCustomValidity(DOMString error);

    [SameObject] readonly attribute NodeList labels;

    void select();
    attribute unsigned long selectionStart;
    attribute unsigned long selectionEnd;
    attribute DOMString selectionDirection;
    void setRangeText(DOMString replacement);
    void setRangeText(DOMString replacement, unsigned long start, unsigned long end, optional
    SelectionMode selectionMode = "preserve");
    void setSelectionRange(unsigned long start, unsigned long end, optional DOMString direction);
};

interface HTMLKeygenElement : HTMLElement {
    attribute boolean autofocus;
    attribute DOMString challenge;
    attribute boolean disabled;
    readonly attribute HTMLFormElement? form;
    attribute DOMString keytype;
    attribute DOMString name;

    readonly attribute DOMString type;

    readonly attribute boolean willValidate;
    readonly attribute ValidityState validity;
    readonly attribute DOMString validationMessage;
    boolean checkValidity();
    boolean reportValidity();
    void setCustomValidity(DOMString error);

    [SameObject] readonly attribute NodeList labels;
};

interface HTMLOutputElement : HTMLElement {
    [SameObject, PutForwards=value] readonly attribute DOMTokenList htmlFor;
    readonly attribute HTMLFormElement? form;
    attribute DOMString name;

```

```
    readonly attribute DOMString type;
    attribute DOMString defaultValue;
    attribute DOMString value;

    readonly attribute boolean willValidate;
    readonly attribute ValidityState validity;
    readonly attribute DOMString validationMessage;
    boolean checkValidity();
    boolean reportValidity();
    void setCustomValidity(DOMString error);

    [SameObject] readonly attribute NodeList labels;
};

interface HTMLProgressElement : HTMLElement {
    attribute double value;
    attribute double max;
    readonly attribute double position;
    [SameObject] readonly attribute NodeList labels;
};

interface HTMLMeterElement : HTMLElement {
    attribute double value;
    attribute double min;
    attribute double max;
    attribute double low;
    attribute double high;
    attribute double optimum;
    [SameObject] readonly attribute NodeList labels;
};

interface HTMLFieldSetElement : HTMLElement {
    attribute boolean disabled;
    readonly attribute HTMLFormElement? form;
    attribute DOMString name;

    readonly attribute DOMString type;

    [SameObject] readonly attribute HTMLCollection elements;

    readonly attribute boolean willValidate;
    [SameObject] readonly attribute ValidityState validity;
    readonly attribute DOMString validationMessage;
    boolean checkValidity();
    boolean reportValidity();
    void setCustomValidity(DOMString error);
};

interface HTMLLegendElement : HTMLElement {
    readonly attribute HTMLFormElement? form;
};

enum SelectionMode {
    "select",
    "start",
    "end",
    "preserve" // default
};

interface ValidityState {
    readonly attribute boolean valueMissing;
    readonly attribute boolean typeMismatch;
    readonly attribute boolean patternMismatch;
    readonly attribute boolean tooLong;
```

```
    readonly attribute boolean tooShort;
    readonly attribute boolean rangeUnderflow;
    readonly attribute boolean rangeOverflow;
    readonly attribute boolean stepMismatch;
    readonly attribute boolean badInput;
    readonly attribute boolean customError;
    readonly attribute boolean valid;
};

interface HTMLDetailsElement : HTMLElement {
    attribute boolean open;
};

interface HTMLMenuElement : HTMLElement {
    attribute DOMString type;
    attribute DOMString label;
};

interface HTMLMenuItemElement : HTMLElement {
    attribute DOMString type;
    attribute DOMString label;
    attribute DOMString icon;
    attribute boolean disabled;
    attribute boolean checked;
    attribute DOMString radiogroup;
    attribute boolean default;
};

[Constructor(DOMString type, optional RelatedEventInit eventInitDict)]
interface RelatedEvent : Event {
    readonly attribute EventTarget? relatedTarget;
};

dictionary RelatedEventInit : EventInit {
    EventTarget? relatedTarget;
};

interface HTMLScriptElement : HTMLElement {
    attribute DOMString src;
    attribute DOMString type;
    attribute DOMString charset;
    attribute boolean async;
    attribute boolean defer;
    attribute DOMString? crossOrigin;
    attribute DOMString text;
    attribute DOMString nonce;
};

interface HTMLTemplateElement : HTMLElement {
    readonly attribute DocumentFragment content;
};

typedef (CanvasRenderingContext2D or WebGLRenderingContext) RenderingContext;

interface HTMLCanvasElement : HTMLElement {
    attribute unsigned long width;
    attribute unsigned long height;

    RenderingContext? getContext(DOMString contextId, any... arguments);
    boolean probablySupportsContext(DOMString contextId, any... arguments);

    DOMString toDataURL(optional DOMString type, any... arguments);
    void toBlob(BlobCallback _callback, optional DOMString type, any... arguments);
};
```

```

callback BlobCallback = void (Blob? blob);

[NoInterfaceObject]
interface ElementContentEditable {
  attribute DOMString contentEditable;
  readonly attribute boolean isContentEditable;
};

interface DataTransfer {
  attribute DOMString dropEffect;
  attribute DOMString effectAllowed;

  [SameObject] readonly attribute DataTransferItemList items;

  void setDragImage(Element image, long x, long y);

  /* old interface */
  [SameObject] readonly attribute DOMString[] types;
  DOMString getData(DOMString format);
  void setData(DOMString format, DOMString data);
  void clearData(optional DOMString format);
  [SameObject] readonly attribute FileList files;
};

interface DataTransferItemList {
  readonly attribute unsigned long length;
  getter DataTransferItem (unsigned long index);
  DataTransferItem? add(DOMString data, DOMString type);
  DataTransferItem? add(File data);
  void remove(unsigned long index);
  void clear();
};

interface DataTransferItem {
  readonly attribute DOMString kind;
  readonly attribute DOMString type;
  void getAsString(FunctionStringCallback? _callback);
  File? getAsFile();
};

callback FunctionStringCallback = void (DOMString data);

[Constructor(DOMString type, optional DragEventInit eventInitDict)]
interface DragEvent : MouseEvent {
  readonly attribute DataTransfer? dataTransfer;
};

dictionary DragEventInit : MouseEventInit {
  DataTransfer? dataTransfer = null;
};

[PrimaryGlobal, LegacyUnenumerableNamedProperties]
/*sealed*/ interface Window : EventTarget {
  // the current browsing context
  [Unforgeable] readonly attribute WindowProxy window;
  [Replaceable] readonly attribute WindowProxy self;
  [Unforgeable] readonly attribute Document document;
  attribute DOMString name;
  [PutForwards=href, Unforgeable] readonly attribute Location location;
  readonly attribute History history;
  [Replaceable] readonly attribute BarProp locationbar;
  [Replaceable] readonly attribute BarProp menubar;
  [Replaceable] readonly attribute BarProp personalbar;

```

```

[Replaceable] readonly attribute BarProp scrollbars;
[Replaceable] readonly attribute BarProp statusbar;
[Replaceable] readonly attribute BarProp toolbar;
attribute DOMString status;
void close();
readonly attribute boolean closed;
void stop();
void focus();
void blur();

// other browsing contexts
[Replaceable] readonly attribute WindowProxy frames;
[Replaceable] readonly attribute unsigned long length;
[Unforgeable] readonly attribute WindowProxy top;
attribute any opener;
[Replaceable] readonly attribute WindowProxy parent;
readonly attribute Element? frameElement;
WindowProxy open(optional DOMString url = "about:blank", optional DOMString target = "_blank",
[NotNullAs=EmptyString] optional DOMString features = "", optional boolean replace = false);
getter WindowProxy (unsigned long index);
getter object (DOMString name);

// the user agent
readonly attribute Navigator navigator;

// user prompts
void alert();
void alert(DOMString message);
boolean confirm(optional DOMString message = "");
DOMString? prompt(optional DOMString message = "", optional DOMString default = "");
void print();
any showModalDialog(DOMString url, optional any argument); // deprecated

unsigned long requestAnimationFrame(FrameRequestCallback callback);
void cancelAnimationFrame(unsigned long handle);
};
Window implements GlobalEventHandlers;
Window implements WindowEventHandlers;

callback FrameRequestCallback = void (DOMHighResTimeStamp time);

interface BarProp {
  readonly attribute boolean visible;
};

enum ScrollRestoration { "auto", "manual" };

interface History {
  readonly attribute unsigned long length;
  attribute ScrollRestoration scrollRestoration;
  readonly attribute any state;
  void go(optional long delta = 0);
  void back();
  void forward();
  void pushState(any data, DOMString title, optional DOMString? url = null);
  void replaceState(any data, DOMString title, optional DOMString? url = null);
};

[Unforgeable]
interface Location {
  stringifier attribute USVString href;
  readonly attribute USVString origin;
  attribute USVString protocol;
  attribute USVString host;

```



```
    attribute USVString hostname;
    attribute USVString port;
    attribute USVString pathname;
    attribute USVString search;
    attribute USVString hash;

    void assign(USVString url);
    void replace(USVString url);
    void reload();

    [SameObject] readonly attribute USVString[] ancestorOrigins;
};

[Constructor(DOMString type, optional PopStateEventInit eventInitDict), Exposed=(Window,Worker)]
interface PopStateEvent : Event {
    readonly attribute any state;
};

dictionary PopStateEventInit : EventInit {
    any state;
};

[Constructor(DOMString type, optional HashChangeEventInit eventInitDict), Exposed=
(Window,Worker)]
interface HashChangeEvent : Event {
    readonly attribute DOMString oldURL;
    readonly attribute DOMString newURL;
};

dictionary HashChangeEventInit : EventInit {
    DOMString oldURL;
    DOMString newURL;
};

[Constructor(DOMString type, optional PageTransitionEventInit eventInitDict), Exposed=
(Window,Worker)]
interface PageTransitionEvent : Event {
    readonly attribute boolean persisted;
};

dictionary PageTransitionEventInit : EventInit {
    boolean persisted;
};

interface BeforeUnloadEvent : Event {
    attribute DOMString returnValue;
};

[NoInterfaceObject, Exposed=(Window, Worker)]
interface NavigatorOnLine {
    readonly attribute boolean onLine;
};

[Constructor(DOMString type, optional ErrorEventInit eventInitDict), Exposed=(Window, Worker)]
interface ErrorEvent : Event {
    readonly attribute DOMString message;
    readonly attribute DOMString filename;
    readonly attribute unsigned long lineno;
    readonly attribute unsigned long colno;
    readonly attribute any error;
};

dictionary ErrorEventInit : EventInit {
    DOMString message = "";
};
```

```

    DOMString filename = "";
    unsigned long lineno = 0;
    unsigned long colno = 0;
    any error = null;
};

[Constructor(DOMString type, PromiseRejectionEventInit eventInitDict), Exposed=(Window,Worker)]
interface PromiseRejectionEvent : Event {
    readonly attribute Promise<any> promise;
    readonly attribute any reason;
};

dictionary PromiseRejectionEventInit : EventInit {
    required Promise<any> promise;
    any reason;
};

[TreatNonObjectAsNull]
callback EventHandlerNonNull = any = any (Event event);
typedef EventHandlerNonNull? EventHandler;

[TreatNonObjectAsNull]
callback OnErrorEventHandlerNonNull = any ((Event or DOMString) event, optional DOMString
source, optional unsigned long lineno, optional unsigned long column, optional any error);
typedef OnErrorEventHandlerNonNull? OnErrorEventHandler;

[TreatNonObjectAsNull]
callback OnBeforeUnloadEventHandlerNonNull = DOMString? (Event event);
typedef OnBeforeUnloadEventHandlerNonNull? OnBeforeUnloadEventHandler;

[NoInterfaceObject]
interface GlobalEventHandlers {
    attribute EventHandler onabort;
    attribute EventHandler onblur;
    attribute EventHandler oncancel;
    attribute EventHandler oncanplay;
    attribute EventHandler oncanplaythrough;
    attribute EventHandler onchange;
    attribute EventHandler onclick;
    attribute EventHandler onclose;
    attribute EventHandler oncontextmenu;
    attribute EventHandler oncuechange;
    attribute EventHandler ondblclick;
    attribute EventHandler ondrag;
    attribute EventHandler ondragend;
    attribute EventHandler ondragenter;
    attribute EventHandler ondragexit;
    attribute EventHandler ondragleave;
    attribute EventHandler ondragover;
    attribute EventHandler ondragstart;
    attribute EventHandler ondrop;
    attribute EventHandler ondurationchange;
    attribute EventHandler onemptied;
    attribute EventHandler onended;
    attribute OnErrorEventHandler onerror;
    attribute EventHandler onfocus;
    attribute EventHandler oninput;
    attribute EventHandler oninvalid;
    attribute EventHandler onkeydown;
    attribute EventHandler onkeypress;
    attribute EventHandler onkeyup;
    attribute EventHandler onload;
    attribute EventHandler onloadeddata;
    attribute EventHandler onloadedmetadata;

```

```

    attribute EventHandler onloadstart;
    attribute EventHandler onmousedown;
    [LenientThis] attribute EventHandler onmouseenter;
    [LenientThis] attribute EventHandler onmouseleave;
    attribute EventHandler onmousemove;
    attribute EventHandler onmouseout;
    attribute EventHandler onmouseover;
    attribute EventHandler onmouseup;
    attribute EventHandler onwheel;
    attribute EventHandler onpause;
    attribute EventHandler onplay;
    attribute EventHandler onplaying;
    attribute EventHandler onprogress;
    attribute EventHandler onratechange;
    attribute EventHandler onreset;
    attribute EventHandler onresize;
    attribute EventHandler onscroll;
    attribute EventHandler onseeked;
    attribute EventHandler onseeking;
    attribute EventHandler onselect;
    attribute EventHandler onshow;
    attribute EventHandler onstalled;
    attribute EventHandler onsubmit;
    attribute EventHandler onsuspend;
    attribute EventHandler ontimeupdate;
    attribute EventHandler ontoggle;
    attribute EventHandler onvolumechange;
    attribute EventHandler onwaiting;
};

[NoInterfaceObject]
interface WindowEventHandlers {
    attribute EventHandler onafterprint;
    attribute EventHandler onbeforeprint;
    attribute OnBeforeUnloadEventHandler onbeforeunload;
    attribute EventHandler onhashchange;
    attribute EventHandler onlanguagechange;
    attribute EventHandler onmessage;
    attribute EventHandler onoffline;
    attribute EventHandler ononline;
    attribute EventHandler onpagehide;
    attribute EventHandler onpageshow;
    attribute EventHandler onrejectionhandled;
    attribute EventHandler onpopstate;
    attribute EventHandler onstorage;
    attribute EventHandler onunhandledrejection;
    attribute EventHandler onunload;
};

[NoInterfaceObject]
interface DocumentAndElementEventHandlers {
    attribute EventHandler oncopy;
    attribute EventHandler oncut;
    attribute EventHandler onpaste;
};

[NoInterfaceObject, Exposed=(Window, Worker)]
interface WindowBase64 {
    DOMString btoa(DOMString btoa);
    DOMString atob(DOMString atob);
};
Window implements WindowBase64;
WorkerGlobalScope implements WindowBase64;

```

```

[NoInterfaceObject, Exposed=(Window,Worker)]
interface WindowTimers {
  long setTimeout((Function or DOMString) handler, optional long timeout = 0, any... arguments);
  void clearTimeout(optional long handle = 0);
  long setInterval((Function or DOMString) handler, optional long timeout = 0, any...
arguments);
  void clearInterval(optional long handle = 0);
};
Window implements WindowTimers;
WorkerGlobalScope implements WindowTimers;

[NoInterfaceObject]
interface WindowModal {
  readonly attribute any dialogArguments;
  attribute any returnValue;
};

interface Navigator {
  // objects implementing this interface also implement the interfaces given below
};
Navigator implements NavigatorID;
Navigator implements NavigatorLanguage;
Navigator implements NavigatorOnLine;
Navigator implements NavigatorContentUtils;
Navigator implements NavigatorCookies;
Navigator implements NavigatorPlugins;

[NoInterfaceObject, Exposed=(Window, Worker)]
interface NavigatorID {
  [Exposed=Window] readonly attribute DOMString appName; // constant "Mozilla"
  readonly attribute DOMString appName; // constant "Netscape"
  readonly attribute DOMString appVersion;
  readonly attribute DOMString platform;
  [Exposed=Window]readonly attribute DOMString product; // constant "Gecko"
  readonly attribute DOMString userAgent;
};

[NoInterfaceObject, Exposed=(Window, Worker)]
interface NavigatorLanguage {
  readonly attribute DOMString? language;
  readonly attribute DOMString[] languages;
};

[NoInterfaceObject]
interface NavigatorContentUtils {
  // content handler registration
  void registerProtocolHandler(DOMString scheme, DOMString url, DOMString title);
  void unregisterProtocolHandler(DOMString scheme, DOMString url);
};

[NoInterfaceObject]
interface NavigatorCookies {
  readonly attribute boolean cookieEnabled;
};

[NoInterfaceObject]
interface NavigatorPlugins {
  interface NavigatorPlugins {
    [SameObject] readonly attribute PluginArray plugins;
    [SameObject] readonly attribute MimeTypeArray mimeTypes;
    boolean javaEnabled();
  };
};

interface PluginArray {
  void refresh(optional boolean reload = false);
};

```

```

    readonly attribute unsigned long length;
    getter Plugin? item(unsigned long index);
    getter Plugin? namedItem(DOMString name);
};

interface MimeTypeArray {
    readonly attribute unsigned long length;
    getter MimeType? item(unsigned long index);
    getter MimeType? namedItem(DOMString name);
};

interface Plugin {
    readonly attribute DOMString name;
    readonly attribute DOMString description;
    readonly attribute DOMString filename;
    readonly attribute unsigned long length;
    getter MimeType? item(unsigned long index);
    getter MimeType? namedItem(DOMString name);
};

interface MimeType {
    readonly attribute DOMString type;
    readonly attribute DOMString description;
    readonly attribute DOMString suffixes; // comma-separated
    readonly attribute Plugin enabledPlugin;
};

[Exposed=(Window, Worker)]
interface ImageBitmap {
    readonly attribute unsigned long width;
    readonly attribute unsigned long height;
};

typedef (HTMLImageElement or
        HTMLVideoElement or
        HTMLCanvasElement or
        Blob or
        ImageData or
        CanvasRenderingContext2D or
        ImageBitmap) ImageBitmapSource;

[NoInterfaceObject, Exposed=(Window, Worker)]
interface ImageBitmapFactories {
    Promise<ImageBitmap> createImageBitmap(ImageBitmapSource image);
    Promise<ImageBitmap> createImageBitmap(ImageBitmapSource image, long sx, long sy, long sw,
    long sh);
};
Window implements ImageBitmapFactories;
WorkerGlobalScope implements ImageBitmapFactories;

interface HTMLAppletElement : HTMLElement {
    attribute DOMString align;
    attribute DOMString alt;
    attribute DOMString archive;
    attribute DOMString code;
    attribute DOMString codeBase;
    attribute DOMString height;
    attribute unsigned long hspace;
    attribute DOMString name;
    attribute DOMString _object; // the underscore is not part of the identifier
    attribute unsigned long vspace;
    attribute DOMString width;
};

```

```
interface HTMLMarqueeElement : HTMLElement {
    attribute DOMString behavior;
    attribute DOMString bgColor;
    attribute DOMString direction;
    attribute DOMString height;
    attribute unsigned long hspace;
    attribute long loop;
    attribute unsigned long scrollAmount;
    attribute unsigned long scrollDelay;
    attribute boolean trueSpeed;
    attribute unsigned long vspace;
    attribute DOMString width;

    attribute EventHandler onbounce;
    attribute EventHandler onfinish;
    attribute EventHandler onstart;

    void start();
    void stop();
};

interface HTMLFrameSetElement : HTMLElement {
    attribute DOMString cols;
    attribute DOMString rows;
};
HTMLFrameSetElement implements WindowEventHandlers;

interface HTMLFrameElement : HTMLElement {
    attribute DOMString name;
    attribute DOMString scrolling;
    attribute DOMString src;
    attribute DOMString frameBorder;
    attribute boolean noResize;
    readonly attribute Document? contentDocument;
    readonly attribute WindowProxy? contentWindow;

    [TreatNullAs=EmptyString] attribute DOMString marginHeight;
    [TreatNullAs=EmptyString] attribute DOMString marginWidth;
};

[Exposed=(Window, SharedWorker)]
interface ApplicationCache : EventTarget {
    // update status
    const unsigned short UNCACHED = 0;
    const unsigned short IDLE = 1;
    const unsigned short CHECKING = 2;
    const unsigned short DOWNLOADING = 3;
    const unsigned short UPDATEREADY = 4;
    const unsigned short OBSOLETE = 5;
    readonly attribute unsigned short status;

    // updates
    void update();
    void abort();
    void swapCache();

    // events
    attribute EventHandler onchecking;
    attribute EventHandler onerror;
    attribute EventHandler onnoupdate;
    attribute EventHandler ondownloading;
    attribute EventHandler onprogress;
    attribute EventHandler onupdateready;
    attribute EventHandler oncached;
```

```
    attribute EventHandler onobsolete;
};

partial interface HTMLAnchorElement {
    attribute DOMString coords;
    attribute DOMString charset;
    attribute DOMString name;
    attribute DOMString shape;
};

partial interface HTMLAreaElement {
    attribute boolean noHref;
};

partial interface HTMLBodyElement {
    [TreatNullAs=EmptyString] attribute DOMString text;
    [TreatNullAs=EmptyString] attribute DOMString link;
    [TreatNullAs=EmptyString] attribute DOMString vLink;
    [TreatNullAs=EmptyString] attribute DOMString aLink;
    [TreatNullAs=EmptyString] attribute DOMString bgColor;
    attribute DOMString background;
};

partial interface HTMLBRElement {
    attribute DOMString clear;
};

partial interface HTMLTableCaptionElement {
    attribute DOMString align;
};

partial interface HTMLTableColElement {
    attribute DOMString align;
    attribute DOMString ch;
    attribute DOMString chOff;
    attribute DOMString vAlign;
    attribute DOMString width;
};

interface HTMLDirectoryElement : HTMLElement {
    attribute boolean compact;
};

partial interface HTMLDivElement {
    attribute DOMString align;
};

partial interface HTMLDListElement {
    attribute boolean compact;
};

partial interface HTMLEmbedElement {
    attribute DOMString align;
    attribute DOMString name;
};

interface HTMLFontElement : HTMLElement {
    [TreatNullAs=EmptyString] attribute DOMString color;
    attribute DOMString face;
    attribute DOMString size;
};

partial interface HTMLHeadingElement {
    attribute DOMString align;
```

```
};

partial interface HTMLHRElement {
    attribute DOMString align;
    attribute DOMString color;
    attribute boolean noShade;
    attribute DOMString size;
    attribute DOMString width;
};

partial interface HTMLHtmlElement {
    attribute DOMString version;
};

partial interface HTMLIFrameElement {
    attribute DOMString align;
    attribute DOMString scrolling;
    attribute DOMString frameBorder;

    [TreatNullAs=EmptyString] attribute DOMString marginHeight;
    [TreatNullAs=EmptyString] attribute DOMString marginWidth;
};

partial interface HTMLImageElement {
    attribute DOMString name;
    attribute DOMString lowsrc;
    attribute DOMString align;
    attribute unsigned long hspace;
    attribute unsigned long vspace;

    [TreatNullAs=EmptyString] attribute DOMString border;
};

partial interface HTMLInputElement {
    attribute DOMString align;
    attribute DOMString useMap;
};

partial interface HTMLLegendElement {
    attribute DOMString align;
};

partial interface HTMLLIElement {
    attribute DOMString type;
};

partial interface HTMLLinkElement {
    attribute DOMString charset;
    attribute DOMString target;
};

partial interface HTMLMenuElement {
    attribute boolean compact;
};

partial interface HTMLMetaElement {
    attribute DOMString scheme;
};

partial interface HTMLObjectElement {
    attribute DOMString align;
    attribute DOMString archive;
    attribute DOMString code;
    attribute boolean declare;
```



```
    attribute unsigned long hspace;
    attribute DOMString standby;
    attribute unsigned long vspace;
    attribute DOMString codeBase;
    attribute DOMString codeType;
    attribute DOMString useMap;

    [TreatNullAs=EmptyString] attribute DOMString border;
};

partial interface HTMLListElement {
    attribute boolean compact;
};

partial interface HTMLParagraphElement {
    attribute DOMString align;
};

partial interface HTMLParamElement {
    attribute DOMString type;
    attribute DOMString valueType;
};

partial interface HTMLPreElement {
    attribute long width;
};

partial interface HTMLScriptElement {
    attribute DOMString event;
    attribute DOMString htmlFor;
};

partial interface HTMLTableElement {
    attribute DOMString align;
    attribute DOMString border;
    attribute DOMString frame;
    attribute DOMString rules;
    attribute DOMString summary;
    attribute DOMString width;

    [TreatNullAs=EmptyString] attribute DOMString bgColor;
    [TreatNullAs=EmptyString] attribute DOMString cellPadding;
    [TreatNullAs=EmptyString] attribute DOMString cellSpacing;
};

partial interface HTMLTableSectionElement {
    attribute DOMString align;
    attribute DOMString ch;
    attribute DOMString chOff;
    attribute DOMString vAlign;
};

partial interface HTMLTableCellElement {
    attribute DOMString align;
    attribute DOMString axis;
    attribute DOMString height;
    attribute DOMString width;

    attribute DOMString ch;
    attribute DOMString chOff;
    attribute boolean noWrap;
    attribute DOMString vAlign;

    [TreatNullAs=EmptyString] attribute DOMString bgColor;
```

```

};

partial interface HTMLTableDataCellElement {
    attribute DOMString abbr;
};

partial interface HTMLTableRowElement {
    attribute DOMString align;
    attribute DOMString ch;
    attribute DOMString chOff;
    attribute DOMString vAlign;

    [TreatNullAs=EmptyString] attribute DOMString bgColor;
};

partial interface HTMLULListElement {
    attribute boolean compact;
    attribute DOMString type;
};

partial interface Document {
    [TreatNullAs=EmptyString] attribute DOMString fgColor;
    [TreatNullAs=EmptyString] attribute DOMString linkColor;
    [TreatNullAs=EmptyString] attribute DOMString vlinkColor;
    [TreatNullAs=EmptyString] attribute DOMString alinkColor;
    [TreatNullAs=EmptyString] attribute DOMString bgColor;

    [SameObject] readonly attribute HTMLCollection anchors;
    [SameObject] readonly attribute HTMLCollection applets;

    void clear();
    void captureEvents();
    void releaseEvents();

    [SameObject] readonly attribute HTMLAllCollection all;
};

partial interface Window {
    void captureEvents();
    void releaseEvents();

    [Replaceable, SameObject] readonly attribute External external;
};

[NoInterfaceObject]
interface External {
    void AddSearchProvider();
    void IsSearchProviderInstalled();
};

```

§ References

§ Normative References

[ABNF]

D. Crocker, Ed.; P. Overell. [Augmented BNF for Syntax Specifications: ABNF](https://tools.ietf.org/html/rfc5234). January 2008. Internet Standard. URL: <https://tools.ietf.org/html/rfc5234>

[BCP47]

A. Phillips; M. Davis. [Tags for Identifying Languages](https://tools.ietf.org/html/bcp47). September 2009. IETF Best Current Practice. URL: <https://tools.ietf.org/html/bcp47>

[BIDI]

Mark Davis; Aharon Lanin; Andrew Glass. [Unicode Bidirectional Algorithm](#). 5 June 2014. Unicode Standard Annex #9. URL: <http://www.unicode.org/reports/tr9/>

[CANVAS-2D]

Rik Cabanier; et al. [HTML Canvas 2D Context](#). 19 November 2015. REC. URL: <https://www.w3.org/TR/2dcontext/>

[CLDR]

[Unicode Common Locale Data Repository](#). URL: <http://cldr.unicode.org/>

[COOKIES]

A. Barth. [HTTP State Management Mechanism](#). April 2011. Proposed Standard. URL: <https://tools.ietf.org/html/rfc6265>

[CSP3]

Mike West. [Content Security Policy Level 3](#). 1 September 2016. WD. URL: <https://www.w3.org/TR/CSP3/>

[CSS-2015]

Tab Atkins Jr.; Erika Etemad; Florian Rivoal. [CSS Snapshot 2015](#). 13 October 2015. NOTE. URL: <https://www.w3.org/TR/css-2015/>

[CSS-COLOR-4]

Tab Atkins Jr.; Chris Lilley. [CSS Color Module Level 4](#). 5 July 2016. WD. URL: <https://www.w3.org/TR/css-color-4/>

[CSS-DISPLAY-3]

Tab Atkins Jr.; Erika Etemad. [CSS Display Module Level 3](#). 15 October 2015. WD. URL: <https://www.w3.org/TR/css-display-3/>

[CSS-FONT-LOADING-3]

Tab Atkins Jr.. [CSS Font Loading Module Level 3](#). 22 May 2014. LCWD. URL: <https://www.w3.org/TR/css-font-loading-3/>

[CSS-FONTS-3]

John Daggett. [CSS Fonts Module Level 3](#). 3 October 2013. CR. URL: <https://www.w3.org/TR/css-fonts-3/>

[CSS-OVERFLOW-3]

David Baron; Florian Rivoal. [CSS Overflow Module Level 3](#). 31 May 2016. WD. URL: <https://www.w3.org/TR/css-overflow-3/>

[CSS-SIZING-3]

Tab Atkins Jr.; Erika Etemad. [CSS Intrinsic & Extrinsic Sizing Module Level 3](#). 12 May 2016. WD. URL: <https://www.w3.org/TR/css-sizing-3/>

[CSS-STYLE-ATTR]

Tantek Çelik; Erika Etemad. [CSS Style Attributes](#). 7 November 2013. REC. URL: <https://www.w3.org/TR/css-style-attr>

[CSS-SYNTAX-3]

Tab Atkins Jr.; Simon Sapin. [CSS Syntax Module Level 3](#). 20 February 2014. CR. URL: <https://www.w3.org/TR/css-syntax-3/>

[CSS-TEXT-3]

Erika Etemad; Koji Ishii. [CSS Text Module Level 3](#). 10 October 2013. LCWD. URL: <https://www.w3.org/TR/css-text-3/>

[CSS-TRANSITIONS-1]

CSS Transitions Module Level 1 URL: <https://www.w3.org/TR/css3-transitions/>

[CSS-UI-3]

Tantek Çelik; Florian Rivoal. [CSS Basic User Interface Module Level 3 \(CSS3 UI\)](#). 7 July 2015. CR. URL: <https://www.w3.org/TR/css-ui-3/>

[CSS-UI-4]

Florian Rivoal. [CSS Basic User Interface Module Level 4](#). 22 September 2015. WD. URL: <https://www.w3.org/TR/css->

[ui-4/](#)

[CSS-VALUES]

Tab Atkins Jr.; Erika Etemad. [CSS Values and Units Module Level 3](#). 11 June 2015. CR. URL: <https://www.w3.org/TR/css-values/>

[CSS-WRITING-MODES-3]

Erika Etemad; Koji Ishii. [CSS Writing Modes Level 3](#). 15 December 2015. CR. URL: <https://www.w3.org/TR/css-writing-modes-3/>

[CSS2]

Bert Bos; et al. [Cascading Style Sheets Level 2 Revision 1 \(CSS 2.1\) Specification](#). 7 June 2011. REC. URL: <https://www.w3.org/TR/CSS2>

[CSS22]

Bert Bos. [Cascading Style Sheets Level 2 Revision 2 \(CSS 2.2\) Specification](#). 12 April 2016. WD. URL: <https://www.w3.org/TR/CSS22/>

[CSS3-CONTENT]

Erika Etemad; Dave Cramer. [CSS Generated Content Module Level 3](#). 2 June 2016. WD. URL: <https://www.w3.org/TR/css-content-3/>

[CSS3-IMAGES]

Erika Etemad; Tab Atkins Jr.. [CSS Image Values and Replaced Content Module Level 3](#). 17 April 2012. CR. URL: <https://www.w3.org/TR/css3-images/>

[CSS3-RUBY]

Erika Etemad; Koji Ishii. [CSS Ruby Layout Module Level 1](#). 5 August 2014. WD. URL: <https://www.w3.org/TR/css-ruby-1/>

[CSS3-SELECTORS]

Tantek Çelik; et al. [Selectors Level 3](#). 29 September 2011. REC. URL: <https://www.w3.org/TR/css3-selectors/>

[CSS3COLOR]

Tantek Çelik; Chris Lilley; David Baron. [CSS Color Module Level 3](#). 7 June 2011. REC. URL: <https://www.w3.org/TR/css3-color>

[CSSOM]

Simon Pieters; Glenn Adams. [CSS Object Model \(CSSOM\)](#). 17 March 2016. WD. URL: <https://www.w3.org/TR/cssom-1/>

[CSSOM-VIEW]

Simon Pieters. [CSSOM View Module](#). 17 March 2016. WD. URL: <https://www.w3.org/TR/cssom-view-1/>

[DOM]

Anne van Kesteren; et al. [W3C DOM4](#). 19 November 2015. REC. URL: <https://www.w3.org/TR/dom/>

[DOM-Parsing]

Travis Leithead. [DOM Parsing and Serialization](#). 17 May 2016. WD. URL: <https://www.w3.org/TR/DOM-Parsing/>

[ECMA-262]

[ECMAScript Language Specification](#). URL: <https://tc39.github.io/ecma262/>

[ENCODING]

Anne van Kesteren; Joshua Bell; Addison Phillips. [Encoding](#). 20 October 2015. CR. URL: <https://www.w3.org/TR/encoding/>

[EVENTSOURCE]

Ian Hickson. [Server-Sent Events](#). 3 February 2015. REC. URL: <https://www.w3.org/TR/eventsource/>

[FETCH]

Anne van Kesteren. [Fetch Standard](#). Continually Updated Specification. URL: <https://fetch.spec.whatwg.org/>

[FILEAPI]

Arun Ranganathan; Jonas Sicking. [File API](#). 21 April 2015. WD. URL: <https://www.w3.org/TR/FileAPI/>

[FULLSCREEN]

Anne van Kesteren; Tantek Çelik. [Fullscreen](#). 18 November 2014. NOTE. URL: <https://www.w3.org/TR/fullscreen/>

[GEOMETRY-1]

Simon Pieters; Dirk Schulze; Rik Cabanier. [Geometry Interfaces Module Level 1](#). 25 November 2014. CR. URL: <https://www.w3.org/TR/geometry-1/>

[HR-TIME-2]

Ilya Grigorik; James Simonsen; Jatinder Mann. [High Resolution Time Level 2](#). 20 July 2016. WD. URL: <https://www.w3.org/TR/hr-time-2/>

[HTML-AAM-1.0]

Steve Faulkner; et al. [HTML Accessibility API Mappings 1.0](#). 3 December 2015. WD. URL: <https://www.w3.org/TR/html-aam-1.0/>

[HTML-ARIA]

Steve Faulkner. [ARIA in HTML](#). 2 September 2016. WD. URL: <https://www.w3.org/TR/html-aria/>

[HTTP]

HTTP is the union of a set of RFCs:

- [Hypertext Transfer Protocol \(HTTP/1.1\): Message Syntax and Routing \(URL: <https://tools.ietf.org/html/rfc7230>\)](#), R. Fielding, J. Reschke. IETF.
- [Hypertext Transfer Protocol \(HTTP/1.1\): Semantics and Content \(URL: <https://tools.ietf.org/html/rfc7231>\)](#), R. Fielding, J. Reschke. IETF.
- [Hypertext Transfer Protocol \(HTTP/1.1\): Conditional Requests \(URL: <https://tools.ietf.org/html/rfc7232>\)](#), R. Fielding, J. Reschke. IETF.
- [Hypertext Transfer Protocol \(HTTP/1.1\): Range Requests \(URL: <https://tools.ietf.org/html/rfc7233>\)](#), R. Fielding, Y. Lafon, J. Reschke. IETF.
- [Hypertext Transfer Protocol \(HTTP/1.1\): Caching \(URL: <https://tools.ietf.org/html/rfc7234>\)](#), R. Fielding, M. Nottingham, J. Reschke. IETF.
- [Hypertext Transfer Protocol \(HTTP/1.1\): Authentication \(URL: <https://tools.ietf.org/html/rfc7235>\)](#), R. Fielding, J. Reschke. IETF.

[IANAPERMHEADERS]

[Permanent Message Header Field Names](#). IANA.

[ISO3166]

[ISO 3166: Codes for the representation of names of countries and their subdivisions](#). ISO.

[ISO4217]

[ISO 4217: Codes for the representation of currencies and funds](#). ISO.

[JPEG]

Eric Hamilton. [JPEG File Interchange Format](#). September 1992. URL: <https://www.w3.org/Graphics/JPEG/jfif3.pdf>

[MATHML]

Patrick D F Ion; Robert R Miner. [Mathematical Markup Language \(MathML\) 1.01 Specification](#). 7 July 1999. REC. URL: <https://www.w3.org/TR/MathML/>

[MEDIA-FRAGS]

Raphaël Troncy; et al. [Media Fragments URI 1.0 \(basic\)](#). 25 September 2012. REC. URL: <https://www.w3.org/TR/media-frags/>

[MEDIA-SOURCE]

Matthew Wolenetz; et al. [Media Source Extensions](#). 5 July 2016. CR. URL: <https://www.w3.org/TR/media-source/>

[MEDIACAPTURE-STREAMS]

Daniel Burnett; et al. [Media Capture and Streams](https://www.w3.org/TR/mediacapture-streams/). 19 May 2016. CR. URL: <https://www.w3.org/TR/mediacapture-streams/>

[MEDIAQ]

Florian Rivoal; et al. [Media Queries](https://www.w3.org/TR/css3-mediaqueries/). 19 June 2012. REC. URL: <https://www.w3.org/TR/css3-mediaqueries/>

[MEDIAQUERIES-4]

Florian Rivoal; Tab Atkins Jr.. [Media Queries Level 4](https://www.w3.org/TR/mediaqueries-4/). 6 July 2016. WD. URL: <https://www.w3.org/TR/mediaqueries-4/>

[MFREL]

[Microformats Wiki: existing rel values](#). Microformats.

[MIMESNIFF]

NOTE:

As of today the Web community lacks a sufficiently complete, reliable, interoperable, and tested specification for the manner in which content sniffing takes place on the Web. We encourage implementers to exercise caution in this area as the Web community makes progress towards addressing this issue.

Gordon P. Hemsley. [MIME Sniffing Standard](https://mimesniff.spec.whatwg.org/). Continually Updated Specification. URL: <https://mimesniff.spec.whatwg.org/>

[MPEG2TS]

[Information technology -- Generic coding of moving pictures and associated audio information: Systems ITU-T Rec. H.222.0 / ISO/IEC 13818-1:2013](http://www.itu.int/rec/T-REC-H.222.0-201206-I). URL: <http://www.itu.int/rec/T-REC-H.222.0-201206-I>

[MPEG4]

ISO/IEC 14496-12: ISO base media file format. ISO/IEC.

[MPEGDASH]

[ISO/IEC 23009-1:2014 Information technology -- Dynamic adaptive streaming over HTTP \(DASH\) -- Part 1: Media presentation description and segment formats](http://standards.iso.org/ittf/PubliclyAvailableStandards/c065274_ISO_IEC_23009-1_2014.zip). URL: http://standards.iso.org/ittf/PubliclyAvailableStandards/c065274_ISO_IEC_23009-1_2014.zip

[OGGSKELETON]

[Ogg Skeleton 4 Message Headers](http://wiki.xiph.org/SkeletonHeaders). 17 March 2014. URL: <http://wiki.xiph.org/SkeletonHeaders>

[ORIGIN]

A. Barth. [The Web Origin Concept](https://tools.ietf.org/html/rfc6454). December 2011. Proposed Standard. URL: <https://tools.ietf.org/html/rfc6454>

[PAGE-VISIBILITY]

Jatinder Mann; Arvind Jain. [Page Visibility \(Second Edition\)](https://www.w3.org/TR/page-visibility/). 29 October 2013. REC. URL: <https://www.w3.org/TR/page-visibility/>

[PNG]

Tom Lane. [Portable Network Graphics \(PNG\) Specification \(Second Edition\)](https://www.w3.org/TR/PNG). 10 November 2003. REC. URL: <https://www.w3.org/TR/PNG>

[POINTERLOCK]

Vincent Scheib. [Pointer Lock](https://www.w3.org/TR/pointerlock/). 30 August 2016. PR. URL: <https://www.w3.org/TR/pointerlock/>

[PROGRESS-EVENTS]

Anne van Kesteren; Charles McCallie Nevile; Jungkee Song. [Progress Events](https://www.w3.org/TR/progress-events/). 11 February 2014. REC. URL: <https://www.w3.org/TR/progress-events/>

[PROMISES-GUIDE]

Domenic Denicola. [Writing Promise-Using Specifications](https://www.w3.org/2001/tag/doc/promises-guide). 16 February 2016. Finding of the W3C TAG. URL: <https://www.w3.org/2001/tag/doc/promises-guide>

[PSL]

[Public Suffix List](#). Mozilla Foundation.

[RESOURCE-HINTS]

Ilya Grigorik. [Resource Hints](#). 27 May 2016. WD. URL: <https://www.w3.org/TR/resource-hints/>

[RFC1034]

P.V. Mockapetris. [Domain names - concepts and facilities](#). November 1987. Internet Standard. URL: <https://tools.ietf.org/html/rfc1034>

[RFC1123]

R. Braden, Ed.. [Requirements for Internet Hosts - Application and Support](#). October 1989. Internet Standard. URL: <https://tools.ietf.org/html/rfc1123>

[RFC2046]

N. Freed; N. Borenstein. [Multipurpose Internet Mail Extensions \(MIME\) Part Two: Media Types](#). November 1996. Draft Standard. URL: <https://tools.ietf.org/html/rfc2046>

[RFC2119]

S. Bradner. [Key words for use in RFCs to Indicate Requirement Levels](#). March 1997. Best Current Practice. URL: <https://tools.ietf.org/html/rfc2119>

[RFC2318]

H. Lie; B. Bos; C. Lilley. [The text/css Media Type](#). March 1998. Informational. URL: <https://tools.ietf.org/html/rfc2318>

[RFC2397]

L. Masinter. [The "data" URL scheme](#). August 1998. Proposed Standard. URL: <https://tools.ietf.org/html/rfc2397>

[RFC2483]

M. Mealling; R. Daniel. [URI Resolution Services Necessary for URN Resolution](#). January 1999. Experimental. URL: <https://tools.ietf.org/html/rfc2483>

[RFC3279]

L. Bassham; W. Polk; R. Housley. [Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List \(CRL\) Profile](#). April 2002. Proposed Standard. URL: <https://tools.ietf.org/html/rfc3279>

[RFC3447]

J. Jonsson; B. Kaliski. [Public-Key Cryptography Standards \(PKCS\) #1: RSA Cryptography Specifications Version 2.1](#). February 2003. Informational. URL: <https://tools.ietf.org/html/rfc3447>

[RFC4648]

S. Josefsson. [The Base16, Base32, and Base64 Data Encodings](#). October 2006. Proposed Standard. URL: <https://tools.ietf.org/html/rfc4648>

[RFC5280]

D. Cooper; et al. [Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List \(CRL\) Profile](#). May 2008. Proposed Standard. URL: <https://tools.ietf.org/html/rfc5280>

[RFC5322]

P. Resnick, Ed.. [Internet Message Format](#). October 2008. Draft Standard. URL: <https://tools.ietf.org/html/rfc5322>

[RFC5724]

E. Wilde; A. Vaha-Sipila. [URI Scheme for Global System for Mobile Communications \(GSM\) Short Message Service \(SMS\)](#). January 2010. Proposed Standard. URL: <https://tools.ietf.org/html/rfc5724>

[RFC5988]

M. Nottingham. [Web Linking](#). October 2010. Proposed Standard. URL: <https://tools.ietf.org/html/rfc5988>

[RFC6068]

M. Duerst; L. Masinter; J. Zawinski. [The 'mailto' URI Scheme](#). October 2010. Proposed Standard. URL: <https://tools.ietf.org/html/rfc6068>

[RFC6266]

J. Reschke. [Use of the Content-Disposition Header Field in the Hypertext Transfer Protocol \(HTTP\)](#). June 2011. Proposed Standard. URL: <https://tools.ietf.org/html/rfc6266>

[RFC6381]

R. Gellens; D. Singer; P. Frojdh. [The 'Codecs' and 'Profiles' Parameters for "Bucket" Media Types](#). August 2011. Proposed Standard. URL: <https://tools.ietf.org/html/rfc6381>

[RFC6455]

I. Fette; A. Melnikov. [The WebSocket Protocol](#). December 2011. Proposed Standard. URL: <https://tools.ietf.org/html/rfc6455>

[RFC6694]

S. Moonesamy, Ed.. [The "about" URI Scheme](#). August 2012. Informational. URL: <https://tools.ietf.org/html/rfc6694>

[RFC7230]

R. Fielding, Ed.; J. Reschke, Ed.. [Hypertext Transfer Protocol \(HTTP/1.1\): Message Syntax and Routing](#). June 2014. Proposed Standard. URL: <https://tools.ietf.org/html/rfc7230>

[RFC7231]

R. Fielding, Ed.; J. Reschke, Ed.. [Hypertext Transfer Protocol \(HTTP/1.1\): Semantics and Content](#). June 2014. Proposed Standard. URL: <https://tools.ietf.org/html/rfc7231>

[RFC7232]

R. Fielding, Ed.; J. Reschke, Ed.. [Hypertext Transfer Protocol \(HTTP/1.1\): Conditional Requests](#). June 2014. Proposed Standard. URL: <https://tools.ietf.org/html/rfc7232>

[RFC7234]

R. Fielding, Ed.; M. Nottingham, Ed.; J. Reschke, Ed.. [Hypertext Transfer Protocol \(HTTP/1.1\): Caching](#). June 2014. Proposed Standard. URL: <https://tools.ietf.org/html/rfc7234>

[RFC7303]

H. Thompson; C. Lilley. [XML Media Types](#). July 2014. Proposed Standard. URL: <https://tools.ietf.org/html/rfc7303>

[RFC7578]

L. Masinter. [Returning Values from Forms: multipart/form-data](#). July 2015. Proposed Standard. URL: <https://tools.ietf.org/html/rfc7578>

[RFC7595]

D. Thaler, Ed.; T. Hansen; T. Hardie. [Guidelines and Registration Procedures for URI Schemes](#). June 2015. Best Current Practice. URL: <https://tools.ietf.org/html/rfc7595>

[SELECTION-API]

Ryosuke Niwa. [Selection API](#). 21 April 2016. WD. URL: <https://www.w3.org/TR/selection-api/>

[SELECTORS-4]

Selectors Level 4 URL: <https://www.w3.org/TR/selectors4/>

[SERVICE-WORKERS]

Alex Russell; Jungkee Song; Jake Archibald. [Service Workers](#). 25 June 2015. WD. URL: <https://www.w3.org/TR/service-workers/>

[SRGB]

[Amendment 1 - Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB](#). URL: <https://webstore.iec.ch/publication/6168>

[SVG]

Jon Ferraiolo. [Scalable Vector Graphics \(SVG\) 1.0 Specification](#). 4 September 2001. REC. URL: <https://www.w3.org/TR/SVG/>

[SVG11]

Erik Dahlström; et al. [Scalable Vector Graphics \(SVG\) 1.1 \(Second Edition\)](#). 16 August 2011. REC. URL: <https://www.w3.org/TR/SVG11/>

[SVG2]

Nikos Andronikos; et al. [Scalable Vector Graphics \(SVG\) 2](#). 15 September 2015. WD. URL: <https://www.w3.org/TR/SVG2/>

[SVGTiny12]

Ola Andersson; et al. [Scalable Vector Graphics \(SVG\) Tiny 1.2 Specification](#). 22 December 2008. REC. URL: <https://www.w3.org/TR/SVGTiny12/>

[TOUCH-EVENTS]

Doug Schepers; et al. [Touch Events](#). 10 October 2013. REC. URL: <https://www.w3.org/TR/touch-events/>

[UIEVENTS]

Gary Kacmarcik; Travis Leithead. [UI Events](#). 4 August 2016. WD. URL: <https://www.w3.org/TR/uievents/>

[UNICODE]

[The Unicode Standard](#). URL: <http://www.unicode.org/versions/latest/>

[URL]

Note: URLs can be used in numerous different manners, in many differing contexts. For the purpose of producing strict URLs one may wish to consider [\[RFC3986\]](#) [\[RFC3987\]](#). The W3C URL specification defines the term URL, various algorithms for dealing with URLs, and an API for constructing, parsing, and resolving URLs. Developers of Web browsers are advised to keep abreast of the latest URL developments by tracking the progress of <https://url.spec.whatwg.org/>. We expect that the W3C URL draft will evolve along the Recommendation track as the community converges on a definition of URL processing.

Most of the URL-related terms used in the HTML specification (**URL**, **absolute URL**, **relative URL**, **relative schemes**, **scheme component**, **scheme data**, **username**, **password**, **host**, **port**, **path**, **query**, **fragment**, **percent encode**, **get the base**, and **UTF-8 percent encode**) can be straightforwardly mapped to the terminology of [\[RFC3986\]](#) [\[RFC3987\]](#). The `URL` [\[RFC3986\]](#) (formerly known as URL) collection of attributes (e.g. href and protocol) and its required definitions (**input**, **query encoding**, **url**, **update steps**, **set the input**) are considered common practice nowadays. Some of the URL-related terms are still being refined (e.g. **URL parser**, **parse errors**, **URL serializer**, **default encode set**, and **percent decode**).

As a word of caution, there are notable differences in the manner in which Web browsers and other software stacks outside the HTML context handle URLs. While no changes would be accepted to URL processing that would break existing Web content, some important parts of URL processing should therefore be considered as implementation-defined (e.g. parsing file: URLs or operating on URLs that would be syntax errors under the [\[RFC3986\]](#) [\[RFC3987\]](#) syntax).

Anne van Kesteren. [URL Standard](#). Continually Updated Specification. URL: <https://url.spec.whatwg.org/>

[URN]

R. Moats. [URN Syntax](#). May 1997. Proposed Standard. URL: <https://tools.ietf.org/html/rfc2141>

[WAI-ARIA]

James Craig; Michael Cooper; et al. [Accessible Rich Internet Applications \(WAI-ARIA\) 1.0](#). 20 March 2014. REC. URL: <https://www.w3.org/TR/wai-aria/>

[WEBGL]

Chris Marrin (Apple Inc.). [WebGL Specification, Version 1.0](#). 10 February 2011. URL: <https://www.khronos.org/registry/webgl/specs/1.0/>

[WEBIDL]

Cameron McCormack; Boris Zbarsky. [WebIDL Level 1](#). 8 March 2016. CR. URL: <https://www.w3.org/TR/WebIDL-1/>

[WEBM]

[WebM Container Guidelines](#). 26 April 2016. URL: <https://www.webmproject.org/docs/container/>

[WEBSTORAGE]

Ian Hickson. [Web Storage \(Second Edition\)](#). 19 April 2016. REC. URL: <https://www.w3.org/TR/webstorage/>

[WEBWORKERS]

Ian Hickson. [Web Workers](#). 24 September 2015. WD. URL: <https://www.w3.org/TR/workers/>

[X690]

Recommendation X.690 — Information Technology — ASN.1 Encoding Rules — Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER). International Telecommunication

Union.

[XML]

Tim Bray; et al. [Extensible Markup Language \(XML\) 1.0 \(Fifth Edition\)](#). 26 November 2008. REC. URL: <https://www.w3.org/TR/xml>

[XML-NAMES]

Tim Bray; et al. [Namespaces in XML 1.0 \(Third Edition\)](#). 8 December 2009. REC. URL: <https://www.w3.org/TR/xml-names>

[XML-STYLESHEET]

James Clark; Simon Pieters; Henry Thompson. [Associating Style Sheets with XML documents 1.0 \(Second Edition\)](#). 28 October 2010. REC. URL: <https://www.w3.org/TR/xml-styleSheet>

[XMLBASE]

Jonathan Marsh. [XML Base \(Second Edition\)](#). 28 January 2009. REC. URL: <https://www.w3.org/TR/xmlbase/>

[XPATH]

James Clark; Steven DeRose. [XML Path Language \(XPath\) Version 1.0](#). 16 November 1999. REC. URL: <https://www.w3.org/TR/xpath>

[XPTR-XMLNS]

Steven DeRose; et al. [XPointer xmlns\(\) Scheme](#). 25 March 2003. REC. URL: <https://www.w3.org/TR/xptr-xmlns/>

§ Informative References

[APNG]

S. Parmenter; V. Vukicevic; A. Smith. [APNG Specification](#). URL: https://wiki.mozilla.org/APNG_Specification

[ATAG20]

Jan Richards; Jeanne F Spellman; Jutta Treviranus. [Authoring Tool Accessibility Guidelines \(ATAG\) 2.0](#). 24 September 2015. REC. URL: <https://www.w3.org/TR/ATAG20/>

[BATTERY-STATUS]

Anssi Kostiaainen; Mounir Lamouri. [Battery Status API](#). 7 July 2016. CR. URL: <https://www.w3.org/TR/battery-status/>

[BOCU1]

M. Scherer; M. Davis. [UTN #6: BOCU-1: MIME-Compatible Unicode Compression](#). URL: <http://www.unicode.org/notes/tn6/>

[CESU8]

T. Phipps. [UTR #26: Compatibility Encoding Scheme For UTF-16: 8-BIT \(CESU-8\)](#). URL: <http://www.unicode.org/reports/tr26/>

[CHARMOD]

Martin Dürst; et al. [Character Model for the World Wide Web 1.0: Fundamentals](#). 15 February 2005. REC. URL: <https://www.w3.org/TR/charmod/>

[COMPUTABLE]

A. Turing. [On computable numbers, with an application to the Entscheidungsproblem](#), *Proceedings of the London Mathematical Society, series 2, volume 42.*, 1937. URL: <http://www.turingarchive.org/browse.php/B/12>

[CSS-CASCADE-3]

Elika Etemad; Tab Atkins Jr.. [CSS Cascading and Inheritance Level 3](#). 19 May 2016. CR. URL: <https://www.w3.org/TR/css-cascade-3/>

[CSS-IMAGES-4]

CSS Image Values and Replaced Content Module Level 4 URL: <https://www.w3.org/TR/css4-images/>

[CSS-LISTS-3]

Tab Atkins Jr.. [CSS Lists and Counters Module Level 3](#). 20 March 2014. WD. URL: <https://www.w3.org/TR/css-lists-3/>

[CSS3-ANIMATIONS]

Dean Jackson; et al. [CSS Animations](#). 19 February 2013. WD. URL: <https://www.w3.org/TR/css3-animations/>

[CSS3-TRANSITIONS]

Dean Jackson; et al. [CSS Transitions](#). 19 November 2013. WD. URL: <https://www.w3.org/TR/css3-transitions/>

[DOT]

[The DOT Language](#). URL: <http://www.graphviz.org/content/dot-language>

[EDITING]

A. Gregor. [HTML Editing APIs](#). URL: <https://dvcs.w3.org/hg/editing/raw-file/tip/editing.html>

[GIF]

[Graphics Interchange Format](#). 31 July 1990. URL: <https://www.w3.org/Graphics/GIF/spec-gif89a.txt>

[GRAPHICS]

Computer Graphics: Principles and Practice in C, Second Edition, J. Foley, A. van Dam, S. Feiner, J. Hughes. Addison-Wesley. ISBN 0-201-84840-6.

[GREGORIAN]

Inter Gravissimas, A. Lilius, C. Clavius. Gregory XIII Papal Bull, February 1582.

[HTML-POLYGLOT]

Eliot Graff; Leif Halvard Silli. [Polyglot Markup: A robust profile of the HTML5 vocabulary](#). 29 September 2015. NOTE. URL: <https://www.w3.org/TR/html-polyglot/>

[HTML5-DIFF]

Simon Pieters. [HTML5 Differences from HTML4](#). 9 December 2014. NOTE. URL: <https://www.w3.org/TR/html5-diff/>

[INBANDTRACKS]

[Sourcing In-band Media Resource Tracks from Media Containers into HTML](#) (URL: <http://dev.w3.org/html5/html-sourcing-inband-tracks/>), S. Pfeiffer, B. Lund. W3C.

[IndexedDB]

Nikunj Mehta; et al. [Indexed Database API](#). 8 January 2015. REC. URL: <https://www.w3.org/TR/IndexedDB/>

[ISO8601]

Representation of dates and times. International Organization for Standardization. 2004. ISO 8601:2004. URL: http://www.iso.org/iso/catalogue_detail?csnumber=40874

[JLREQ]

Yasuhiro Anan; et al. [Requirements for Japanese Text Layout](#). 3 April 2012. NOTE. URL: <https://www.w3.org/TR/jl-req/>

[MNG]

[MNG \(Multiple-image Network Graphics\) Format](#). G. Randers-Pehrson.

[NPAPI]

[Gecko Plugin API Reference](#). Mozilla.

[OPENSEARCH]

[Autodiscovery in HTML/XHTML](#). In *OpenSearch 1.1 Draft 4*, Section 4.6.2. OpenSearch.org.

[PDF]

[Document management — Portable document format — Part 1: PDF](#). ISO.

[PPUTF8]

[The Properties and Promises of UTF-8](#), M. Dürst. University of Zürich. In *Proceedings of the 11th International Unicode Conference*.

[RFC2152]

D. Goldsmith; M. Davis. [UTF-7 A Mail-Safe Transformation Format of Unicode](#). May 1997. Informational. URL: <https://tools.ietf.org/html/rfc2152>

[RFC3676]

R. Gellens. [The Text/Plain Format and DelSp Parameters](#). February 2004. Proposed Standard. URL: <https://tools.ietf.org/html/rfc3676>

[RFC4287]

M. Nottingham, Ed.; R. Sayre, Ed.. [The Atom Syndication Format](#). December 2005. Proposed Standard. URL: <https://tools.ietf.org/html/rfc4287>

[RFC4329]

B. Hoehrmann. [Scripting Media Types](#). April 2006. Informational. URL: <https://tools.ietf.org/html/rfc4329>

[RUBY-UC]

Richard Ishida. [Use Cases & Exploratory Approaches for Ruby Markup](#). 8 October 2013. NOTE. URL: <https://www.w3.org/TR/ruby-use-cases/>

[SCSU]

[UTR #6: A Standard Compression Scheme For Unicode](#), M. Wolf, K. Whistler, C. Wicksteed, M. Davis, A. Freytag, M. Scherer. Unicode Consortium.

[TIMEZONE]

Addison Phillips; et al. [Working with Time Zones](#). 5 July 2011. NOTE. URL: <https://www.w3.org/TR/timezone>

[TOR]

[Tor](#).

[TZDATABASE]

[Time Zone Database](#). IANA.

[UAAG20]

James Allan; et al. [User Agent Accessibility Guidelines \(UAAG\) 2.0](#). 15 December 2015. NOTE. URL: <https://www.w3.org/TR/UAAG20/>

[UNDO]

Ryosuke Niwa. [UndoManager and DOM Transaction](#). ED. URL: <https://dvcs.w3.org/hg/undomanager/raw-file/tip/undomanager.html>

[UNICODE-SECURITY]

Mark Davis; Michel Suignard. [Unicode Security Considerations](#). URL: <http://www.unicode.org/reports/tr36/>

[UNIVCHARDET]

[A composite approach to language/encoding detection](#), S. Li, K. Momoi. Netscape. In *Proceedings of the 19th International Unicode Conference*.

[UTF8DET]

[Multilingual form encoding](#), M. Dürst. W3C.

[WAI-ARIA-1.1]

Joanmarie Diggs; et al. [Accessible Rich Internet Applications \(WAI-ARIA\) 1.1](#). 21 July 2016. WD. URL: <https://www.w3.org/TR/wai-aria-1.1/>

[WCAG20]

Ben Caldwell; et al. [Web Content Accessibility Guidelines \(WCAG\) 2.0](#). 11 December 2008. REC. URL: <https://www.w3.org/TR/WCAG20/>

[WEBVTT]

Simon Pieters. [WebVTT: The Web Video Text Tracks Format](#). 8 December 2015. WD. URL: <https://www.w3.org/TR/webvtt1/>

[WHATWGWIKI]

[The WHATWG Wiki](#). WHATWG.

[XHR]

Anne van Kesteren. [XMLHttpRequest Standard](#). Continually Updated Specification. URL: <https://xhr.spec.whatwg.org/>

[XML-ENTITY-NAMES]

David Carlisle; Patrick D F Ion. [XML Entity Definitions for Characters \(2nd Edition\)](#). 10 April 2014. REC. URL: <https://www.w3.org/TR/xml-entity-names/>

[XSLT]

James Clark. [XSL Transformations \(XSLT\) Version 1.0](#). 16 November 1999. REC. URL: <https://www.w3.org/TR/xslt>

§ Changes

Changes since [HTML 5.1](#)

This section summarises changes between this specification and [\[HTML 5.1\]](#).

§ Errata corrected

[HTML4 system identifiers are accidentally non-conforming in HTML5.1](#)

Restores the system identifiers in the [HTML syntax](#) section, to use http:// instead of https://.

Changes since [HTML 5.0](#)

This section summarises substantial substantive changes between this specification and [\[HTML5\]](#).

§ Features added

- The [picture](#) and [srcset](#) attributes allow responsive image selection.
- The [details](#) and [summary](#) elements enable authors to provide extended information that users can choose whether to read.
- The [menuitem](#) and type="context" attribute value enable authors to add functionality to the browser's context menu.
- The requestAnimationFrame API allows for more efficient animation.
- enqueueJob and nextJob help explain Promise resolution in terms of microtasks.
- The rev attribute for links, primarily to support RDFa (previously defined in HTML 4).
- HTMLMediaElement and srcObject objects.
- Enable cross-origin track and EventSource and cross-origin content for ImageBitmap in [canvas](#).
- event-source-error, event-track-error and event-track-load events for media fetching.
- onrejectionhandled and onunhandledrejection and APIs for tracking promise rejection.
- HTMLTableCaptionElement, HTMLTableSectionElement, HTMLTableRowElement, for HTML table elements.
- history.scrollRestoration to control where a users' view is directed when navigating through their history.
- IDL [SameObject], for some objects that return collections.
- Add "noopener" to rel and window to allow for browsing contexts to be separated.
- nonce attribute on [script](#) and [style](#) to support the use of Content Security Policy.

§ Features removed

- appCache.
- Media Controllers.
- The command API.
- The usemap attribute on [object](#).
- The accessKeyLabel IDL attribute.
- The form attribute is no longer valid for [label](#).

- The multiple attribute on `input type="range"`.
- `hreflang` and `type` attributes on `<area>`.
- The use of nested `<section>` elements each with an `<h1>` to create an outline.
- Special handling of `isindex` in form submission.
- `navigator.yieldForStorageUpdates()` and the Storage mutex.
- Disallow `tfoot` before `tbody`.
- `[Exposed=Window]` for `HTMLHyperLinkElementUtils`, `[Exposed=Window, Worker]` for `DOMStringMap` and `IDLDate`.

§ Changes to existing features

- The `accesskey` takes a single character as a value (as in HTML 4).
- `<header>` and `<footer>` elements can be nested, if each level is within a sectioning element.
- `<option>` elements can be empty.
- The mousewheel event is called `wheel`.
- The `value` attribute of `input type="submit"` is translatable.
- A `<figcaption>` can appear anywhere within a `<figure>`.
- Having `title`, or writing email to a friend does not make an `<img>` missing `alt` conformant.
- The content of `<time>` is phrasing content, or text.
- Blank `alt` on area elements with duplicate `href` attributes is no longer conformant.
- When navigating internally, the next search for a link etc starts from where the navigation moved to.
- `<img>` and related elements support `width="0"`.
- `.tfoot` and `.createTfoot()` always insert at the end of a table
- `fieldset` and `namedItem` make `HTMLCollections` not `HTMLFormControlsCollections` and `HTMLOptionsCollections`.
- `frameElement` can return `null`.
- For images which don't resolve `currentSrc` is the URL given to the resolver, not necessarily the absolute URL.
- `script` IDL attributes reflect.
- `meta refresh` allows `;` or `url=` to be optional.
- `navigator.javaEnabled()` is a method.
- `fileCallback` is called `blobcallback`.
- The `toBlob()` callback is non-nullable.
- `origin` on `HTMLHyperlinkElementUtils` and `Location` is readonly.
- The first `title` child of an SVG is its title, not the last.
- `window.open()` can return `null`

§ Acknowledgements

Thanks to Tim Berners-Lee for inventing HTML, without which none of this would exist, Dan Connolly, the many who worked to standardise HTML over the last couple of decades or so, and the many more who worked on ideas subsequently incorporated into HTML.

For inestimable work, and the drive to keep HTML up to date, particular thanks are due to Ian Hickson, and the other editors of the WHATWG: Anne van Kesteren, Domenic Denicola, Philip Jägenstedt, Simon Pieters.

Thanks to the participants of the [Responsive Images Community Group](#) and the WHATWG for helping to develop the `<picture>` element, the `srcset` attribute, and the `sizes` attribute. Special thanks to Bruce Lawson for originally suggesting, Theresa O'Connor and Ian Hickson for writing the original `srcset` specification, and Adrian Bateman for providing the group

with guidance. Contributions also from: David Newton, Ilya Grigorik, John Schoenick, and Leon de Rijke.

With apologies to people who have undeservedly not been named, thanks to

Aankhen, Aaron Boodman, Aaron Leventhal, Adam Barth, Adam de Boor, Adam Hepton, Adam Klein, Adam Roben, Addison Phillips, Adele Peterson, Adrian Bateman, Adrian Roselli, Adrian Sutton, Agustín Fernández, Aharon (Vladimir) Lanin, Ajai Tirumali, Akatsuki Kitamura, Alan Plum, Alastair Campbell, Alejandro G. Castro, Alex Bishop, Alex Nicolaou, Alex Plescan, Alex Rousskov, Alexander Farkas, Alexander J. Vincent, Alexander Surkov, Alexandre Morgaut, Alexey Feldgendler, Алексей Проскуряков (Alexey Proskuryakov), Alexis Deveria, Alice Boxhall, Allan Clements, Ami Fischman, Amos Jeffries, Anders Carlsson, André E. Veltstra, Andrea Rendine, Andreas, Andreas Kling, Andrei Popescu, Andres Gomez, Andrew Barfield, Andrew Clover, Andrew Gove, Andrew Grieve, Andrew Oakley, Andrew Sidwell, Andrew Simons, Andrew Smith, Andrew W. Hagen, Andrey V. Lukyanov, Andry Rendy, Andy Earnshaw, Andy Heydon, Andy Palay, Anjana Vakil, Anna Belle Leiserson, Anthony Boyd, Anthony Bryan, Anthony Hickson, Anthony Ramine, Anthony Ricaud, Antonio Olmo Titos, Antti Koivisto, Arkadiusz Michalski, Arne Thomassen, Aron Spohr, Arphen Lin, Arron Eicholz, Arthur Stolyar, Arun Patole, Aryeh Gregor, Asbjørn Ulsberg, Ashley Gullen, Ashley Sheridan, Atsushi Takayama, Aurelien Levy, Ave Wrigley, Axel Dahmen, B Lingafelter, Bart Humphries, Ben Boyle, Ben Buchanan, Ben Godfrey, Ben Lerner, Ben Leslie, Ben Meadowcroft, Ben Millard, Benjamin Carl Wiley Sittler, Benjamin Hawkes-Lewis, Benoit Ren, Bert Bos, Bijan Parsia, Bil Corry, Bill Mason, Bill McCoy, Billy Wong, Bjartur Thorlacius, Björn Höhrmann, Blake Frantz, Bob Lund, Bob Owen, Bobby Holly, Boris Zbarsky, Brad Fuels, Brad Neuberg, Brad Spencer, Brady Eidson, Brendan Eich, Brenton Simpson, Brett Wilson, Brett Zamir, Brian Blakely, Brian Campbell, Brian Korver, Brian Kuhn, Brian M. Dube, Brian Ryner, Brian Smith, Brian Wilson, Bryan Sullivan, Bruce Bailey, Bruce D'Arcus, Bruce Lawson, Bruce Miller, C. Williams, Cameron McCormack, Cameron Zemek, Cao Yipeng, Carlos Amengual, Carlos Gabriel Cardona, Carlos Perelló Marín, Casey Leask, Cătălin Mariș, Chaals McCathie Nevile, Chao Cai, 윤석찬 (Channy Yun), Charl van Niekerk, Charles Iliya Krempeaux, Charu Pandhi, Chris Apers, Chris Cressman, Chris Evans, Chris Morris, Chris Pearce, Chris Peterson, Chris Weber, Christian Biesinger, Christian Johansen, Christian Schmidt, Christoph Päper, Christophe Dumez, Christopher Aillon, Christopher Ferris, Chriswa, Clark Buehler, Cole Robison, Colin Fine, Collin Jackson, Corey Farwell, Corprew Reed, Craig Cockburn, Csaba Gabor, Csaba Marton, Cynthia Shelly, Dan Brickley, Dan Yoder, Daniel Barclay, Daniel Bratell, Daniel Brooks, Daniel Brumbaugh Keeney, Daniel Cheng, Daniel Davis, Daniel Glazman, Daniel Peng, Daniel Schattenkirchner, Daniel Spång, Daniel Steinberg, Daniel Trebbien, Danny Sullivan, Darin Adler, Darin Fisher, Darxus, Dave Camp, Dave Hodder, Dave Lampton, Dave Singer, Dave Townsend, David Baron, David Bloom, David Bruant, David Carlisle, David E. Cleary, David Egan Evans, David Fink, David Flanagan, David Gerard, David Håsäther, David Hyatt, David I. Lehn, David John Burrowes, David Kendal, David MacDonald, David Matja, David Remahl, David Smith, David Storey, David Vest, David Woolley, DeWitt Clinton, Dean Edridge, Dean Edwards, Debi Orton, Derek Featherstone, Devarshi Pant, Devdatta, Dimitri Glazkov, Dimitry Golubovsky, Dirk Pranke, Dirk Schulze, Dirkjan Ochtman, Divya Manian, Dmitry Titov, dolphinling, Dominic Mazzoni, Dominique Hazaël-Massieux, Don Brutzman, Doron Rosenberg, Doug Kramer, Doug Simpinkson, Drew Wilson, Dylan Barrell, Edmund Lai, Eduard Pascual, Eduardo Vela, Theresa O'Connor, Edward Welbourne, Edward Z. Yang, Ehsan Akhgari, Eira Monstad, Eitan Adler, Eliot Graff, Elisabeth Robson, Elizabeth Castro, Elliott Regan, Elliott Sprehn, Elliottte Harold, Eric Carlson, Eric Casler, Eric Lawrence, Eric Rescorla, Eric Semling, Erik Arvidsson, Erik Rose, Evan Jacobs, Evan Martin, Evan Prodromou, Evan Stade, Evert, fantasai, Felix Sasaki, Francesco Schwarz, Francis Brosnan Blazquez, Franck "Shift" Quélain, François Remy, Frank Barchard, Frank Liberato, Frank Olivier, Fredrik Söderquist, 鵜飼文敏 (Fumitoshi Ukai), Futomi Hatano, Gavin Carothers, Gavin Kistner, Gareth Rees, Gary Kačmarčík, Garrett Smith, Geoff Richards, Geoffrey Garen, Sam Sneddon, Gez Lemon, George Lund, George Orno, Gianmarco Armellini, Giovanni Campagna, Giuseppe Pascale, Glenn Adams, Glenn Maynard, Graham Klyne, Greg Botten, Greg Houston, Greg Wilkins, Gregg Tavares, Gregory J. Rosmaita, Grey, Guilherme Johansson Tramontina, Gytis Jakutonis, Håkon Wium Lie, Habib Virji, Hallvord Reiar Michaelsen Steen, Hans S. Tømmerhalt, Hans Stimer, Harald Alvestrand, Henri Sivonen, Henrik Lied, Henry Mason, Henry Story, Heydon Pickering, Hugh Guiney, Hugh Winkler, Ian Bicking, Ian Clelland, Ian Davis, Ian Devlin, Ian Fette, Ian Kilpatrick, Ido Green, Ignacio Javier, Ivan Enderlin, Ivo Emanuel Gonçalves, J. King, Jacob Davies, Jacques Distler, Jake Verbaten, Jakub Łopuszański, Jakub Wilk, James Craig, James Graham, James Greene, James Justin Harrell, James Kozianski, James M Snell, James Perrett, James Robinson, Jamie Lokier, Jan Molnár, Janusz Majnert, Jan-Klaas Kollhof, Jared Jacobs, Jason Duell, Jason Kersey, Jason Kiss, Jason Lustig, Jason White, Jasper Bryant-Greene, Jasper St. Pierre, Jatinder Mann, Jdsmith3000, Jed Hartman, Jeff Balogh, Jeff Cutsinger, Jeff Schiller, Jeff Walden, Jeffrey Yasskin, Jeffrey Zeldman, 胡慧鋒 (Jennifer Braithwaite), Jens Bannmann, Jens Fendler, Jens Lindström, Jens Meiert, Jer Noble, Jeremy Hustman, Jeremy Keith, Jeremy Orlow, Jerry Smith, Jeroen van der Meer, Jesse Renée Beach, Jian Li, Jim Jewett, Jim Ley, Jim Meehan, Jim Michaels, Jirka Kosek, Jjgod Jiang, João Eiras, Jochen Eisinger, Joe Clark, Joe Gregorio, Joel Spolsky, Joel Verhagen, Johan Herland, John Boyer, John Bussjaeger, John Carpenter, John Daggett, John Fallows, John Foliot, John Harding, John Keiser, John Snyders, John Stockton, John-Mark Bell, Johnny Stenback, Jon Ferraiolo, Jon Gibbins, Jon Gunderson, Jon Ribbins, Jon Perlow, Jonas Sickling, Jonathan Cook, Jonathan Kingston, Jonathan Rees, Jonathan Watt, Jonathan Worent, Jonny

Axelsson, Jordan Tucker, Jorgen Horstink, Jorunn Danielsen Newth, Joseph Kesselman, Joseph Mansfield, Joseph Pecoraro, Josh Aas, Josh Hart, Josh Levenberg, Josh Matthews, Joshua Bell, Joshua Berenhaus, Joshua Randall, Jukka K. Korpela, Jules Clément-Ripoche, Julian Reschke, Julio Lopez, Junkee Song, Jürgen Jeka, Justin Lebar, Justin Novosad, Justin Rogers, Justin Schuh, Justin Sinclair, Ka-Sing Chou, Kai Hendry, 吕康豪 (KangHao Lu), Karl Dubost, Karl Groves, Kartikaya Gupta, Kathy Walton, Keith Hall, Keith Yeung, Kelly Ford, Kelly Norton, Kevin Benson, Kevin Gadd, Kevin Cole, Kinuko Yasuda Kornél Pál, Kornel Lesinski, Kris Northfield, Kristof Zelechovski, Krzysztof Maczyński, 黒澤剛志 (Kurosawa Takeshi), Kyle Barnhart, Kyle Hofmann, Kyle Huey, Léonard Bouchet, Lachlan Hunt, Larry Masinter, Larry Page, Lars Gunther, Lars Solberg, Laura Carlson, Laura Granka, Laura L. Carlson, Laura Wisewell, Laurens Holst, Lawrence Forooghian, Lea Verou, Lee Kowalkowski, Leif Halvard Silli, Leif Kornstaedt, Lenny Domnitser, Leonard Rosenthol, Léonie Watson, Leons Petrazickis, Lobotom Dysmon, Logan, Loune, Łukasz Pilorz, Luke Kenneth Casson Leighton, Maciej Stachowiak, Magnus Kristiansen, Maik Merten, Majid Valipour, Malcolm Rowe, Manu Sporny, Manuel Strehl, Manish Tripathi, Mallory van Achterberg, Marat Talanin, Marc Hoyois, Marcus Bointon, Mark Birbeck, Mark Davis, Mark Miller, Mark Nottingham, Mark Pilgrim, Mark Rogers, Mark Rowe, Mark Schenk, Mark Vickers, Mark Wilton-Jones, Marquish, Martijn Wargers, Martin Atkins, Martin Dürst, Martin Honnen, Martin Janecke, Martin Kutschker, Martin Nilsson, Martin Thomson, Masataka Yakura, Masatoshi Kimura, Matheus Martins, Mathias Bynens, Mathieu Henri, Matias Larsson, Matt Falkenhagen, Matt Garrish, Matt May, Matt Rakow, Matt Schmidt, Matt Wright, Matthew Gregan, Matthew Mastracci, Matthew Noorenberghe, Matthew Raymond, Matthew Thomas, Mattias Waldau, Max Romanschuk, Menachem Salomon, Menno van Slooten, Mia Lipner, Micah Dubinko, Michael "Ratt" Iannarelli, Michael A. Nachbaur, Michael A. Puls II, Michael Carter, Michael Daskalov, Michael Day, Michael Dyck, Michael Enright, Michael Gratton, Michael Nordman, Michael Powers, Michael Rakowski, Michael(tm) Smith, Michael Walmsley, Michal Zalewski, Michel Fortin, Michelangelo De Simone, Michiel Bijl, Michiel van der Blonk, Mihai Şucan, Mihai Parparita, Mike Brown, Mike Dierken, Mike Dixon, Mike Hearn, Mike Schinkel, Mike Shaver, Mikko Rantalainen, Mohamed Zergaoui, Mohammad Al Houssami, Momdo Nakamura, Mounir Lamouri, Mount-root-yy, Ms2ger, Nadia Heninger, Nhan, NARUSE Yui, Neil Deakin, Neil Rashbrook, Neil Soiffer, Nicholas Shanks, Nicholas Stimpson, Nicholas Zakas, Nick Levinson, Nickolay Ponomarev, Nicolas Gallagher, Noah Mendelsohn, Noah Slater, Noel Gordon, Nolan Waite, NoozNooz42, Norbert Lindenberg, Ojan Vafai, Olaf Hoffmann, Olav Junker Kjær, Oldřich Vetešník, Oli Studholme, Oliver Hunt, Oliver Rigby, Olivier Gendrin, Olli Pettay, oSand, Pablo Flouret, Patrick Garies, Patrick H. Lauke, Patrik Persson, Paul Adenot, Paul Cotton, Paul Norman, Per-Erik Brodin, Perry Smith, Peter Beverloo, Peter Karlsson, Peter Kasting, Peter Lemieux, Peter Moulder, Peter Occil, Peter Stark, Peter Van der Beken, Peter Winnberg, Peter-Paul Koch, Phil Pickering, Philip Taylor, Philip TAYLOR, Philippe De Ryck, Prateek Rungta, Pravir Gupta, Prayag Verma, 李普君 (Pujun Li), Rabab Gomaa, Rachid Finge, Rachel White, Rafael Weinstein, Rafał Miłecki, Raj Doshi, Rajas Moonka, Ralf Stoltze, Ralph Giles, Raphael Champeimont, Rebeca Ruiz, Remci Mizkur, Remco, Remy Sharp, Rene Saarsoo, Rene Stach, Ric Hardacre, Rich Clark, Rich Doughty, Richa Rupela, Richard Ishida, Richard Schwerdtfeger, Rigo Wenning, Rikkert Koppes, Rimantas Liubertas, Riona Macnamara, Rob Ennals, Rob Jellinghaus, Rob S, Robert Blaut, Robert Collins, Robert Kieffer, Robert Millan, Robert O'Callahan, Robert Sayre, Robin Berjon, Robin Schaulfer, Rodger Combs, Rodney Rehm, Roland Steiner, Roma Matusевич, Roman Ivanov, Roy Fielding, Ruud Steltenpool, Ryan King, Ryan Rion, Ryosuke Niwa, S. Mike Dierken, Sailesh Panchang, Salvatore Loreto, Sam Dutton, Sam Kuper, Sam Ruby, Sam Weinig, Samuel Bronson, Samy Kamkar, Sander van Lambalgen, Sarven Capadisli, 佐藤雅之 (SATO Masayuki), Scott González, Scott Hess, Sean Fraser, Sean Hayes, Sean Hogan, Sean Knapp, Sebastian Markbåge, Sebastian Schnitzenbaumer, Seth Call, Seth Dillingham, Shannon Moeller, Shanti Rao, Shaun Inman, Shiki Okasaka, Shubheksha Jalan, Sierk Bornemann, Sigbjørn Finne, Sigbjørn Vik, Silver Ghost, Silvia Pfeiffer, Šime Vidas, Simo Sutela, Simon Montagu, Simon Spiegel, skeww, Smylers, Srirama Chandra Sekhar Mogali, Stanton McCandlish, Stefan Götz, Stefan Håkansson, Stefan Haustein, Stefan Santesson, Stefan Schumacher, Stefan Weiss, Steffen Meschkat, Stephane Corlosquet, Stephen Cunliffe, Stephen Ma, Stephen White, Steve Comstock, Steve Runyon, Steven Bennett, Steven Garrity, Steven Tate, Steven Wood, Stewart Brodie, Stuart Ballard, Stuart P Bentley, Stuart Langridge, Stuart Parmenter, Subramanian Peruvemba, Sunava Dutta, Susan Borgrink, Susan Lesch, Sylvain Pasche, T. J. Crowder, Tab Atkins-Bittner, Taiju Tsuiki, Takeshi Kurosawa, Takeshi Yoshino, TanteK Çelik, 田村健人 (TAMURA Kent), Taylor Hunt, Ted Mielczarek, Terrence Wood, Thijs van der Vossen, Thomas Broyer, Thomas Koetter, Thomas O'Connor, Tim Baxter, Tim Altman, Tim Johansson, TJ VanToll, Toby Inkster, Tobi Reif, Todd Moody, Tom Baker, Tom Pike, Tommy Thorsen, Tony Ross, Tooru Fujisawa, Travis Leithead, Trevor Saunders, triple-underscore, Tyler Close, Unor, Victor Carbune, Vipul Snehaddeep Chawathe, Vitya Muhachev, Vladimir Katardjiev, Vladimir Vukićević, voracity, Wakaba, Wayne Carr, Wayne Pollock, Wellington Fernando de Macedo, Wes, Weston Ruter, Wilhelm Joys Andersen, Will Levine, William Chen, William Swanson, Wladimir Palant, Wojciech Mach, Wolfram Kriesing, Xan Gregg, xenotheme, Yang Chen, Yaroslav, Ye-Kui Wang, Yehuda Katz, Yi-An Huang, Yngve Nysaeter Pettersen, Yoav Weiss, Yonathan Randolph, Yuzo Fujishima, Zhenbin Xu, Zoltan Herczeg, and Øistein E. Andersen,

for their useful comments, both large and small, that have led to changes to this specification over the years.

Thanks also to everyone who has ever posted about HTML to their blogs, public mailing lists, or forums, including all the

contributors to the [various W3C HTML WG lists](#) and the [various WHATWG lists](#).

The image in the introduction is based on [a photo](#) by [Wonderlane](#). (CC BY 2.0)

The image of the wolf in the embedded content introduction is based on [a photo](#) by [Barry O'Neill](#). (Public domain)

The image of the kettlebell swing in the embedded content introduction is based on [a photo](#) by [kokkarina](#). (CC0 1.0)

The image of two cute kittens in a basket used in the context menu example is based on [a photo](#) by [Alex G](#). (CC BY 2.0)

The Blue Robot Player sprite used in the canvas demo is based on [a work](#) by [JohnColburn](#). (CC BY-SA 3.0)

The photograph of robot 148 climbing the tower at the FIRST Robotics Competition 2013 Silicon Valley Regional is based on [a work](#) by [Lenore Edman](#). (CC BY 2.0)

Parts of this specification are © Copyright 2004-2014 Apple Inc., Mozilla Foundation, and Opera Software ASA. You are granted a license to use, reproduce and create derivative works of this document.