# Table of contents

1 Introduction ............................................................................................................................................................................. 1
2 Common infrastructure ........................................................................................................................................................................ 2
3 Semantics, structure, and APIs of HTML documents .................................................................................................................. 3
4 The elements of HTML ........................................................................................................................................................................ 4
5 Microdata ....................................................................................................................................................................................... 12
6 User interaction ............................................................................................................................................................................... 12
7 Loading web pages .......................................................................................................................................................................... 13
8 Web application APIs ......................................................................................................................................................................... 15
9 Communication ................................................................................................................................................................................... 16
10 Web workers .................................................................................................................................................................................... 17
11 Worklets ............................................................................................................................................................................................ 18
12 Web storage ..................................................................................................................................................................................... 18
13 The HTML syntax ............................................................................................................................................................................ 18
14 The XML syntax ............................................................................................................................................................................... 21
15 Rendering ....................................................................................................................................................................................... 21
16 Obsolete features ............................................................................................................................................................................ 22
17 IANA considerations ....................................................................................................................................................................... 22
Index ........................................................................................................................................................................................................... 23
References .......................................................................................................................................................................................... 23
Acknowledgments .............................................................................................................................................................................. 23
Intellectual property rights ............................................................................................................................................................. 23

## Full table of contents

1 Introduction .................................................................................................................................................................................. 24
1.1 Where does this specification fit? ............................................................................................................................................... 24
1.2 Is this HTML5? ........................................................................................................................................................................... 24
1.3 Background ............................................................................................................................................................................... 25
1.4 Audience .................................................................................................................................................................................... 25
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.1</td>
<td>The Document object</td>
<td>115</td>
</tr>
<tr>
<td>3.1.2</td>
<td>The DocumentOrShadowRoot interface</td>
<td>116</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Resource metadata management</td>
<td>116</td>
</tr>
<tr>
<td>3.1.4</td>
<td>DOM tree accessors</td>
<td>118</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Semantics</td>
<td>122</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.2.2 Elements in the DOM ................................................................................................................. 124
3.2.3 HTML element constructors ....................................................................................................... 126
3.2.4 Element definitions .......................................................................................................................... 129
  3.2.4.1 Attributes ..................................................................................................................................... 130
3.2.5 Content models .................................................................................................................................. 130
  3.2.5.1 The "nothing" content model ...................................................................................................... 131
  3.2.5.2 Kinds of content ......................................................................................................................... 131
    3.2.5.2.1 Metadata content .................................................................................................................. 132
    3.2.5.2.2 Flow content .......................................................................................................................... 132
    3.2.5.2.3 Sectioning content ................................................................................................................. 132
    3.2.5.2.4 Heading content ..................................................................................................................... 133
    3.2.5.2.5 Phrasing content ..................................................................................................................... 133
    3.2.5.2.6 Embedded content .................................................................................................................. 133
    3.2.5.2.7 Interactive content .................................................................................................................. 133
    3.2.5.2.8 Palpable content ..................................................................................................................... 134
    3.2.5.2.9 Script-supporting elements .................................................................................................... 134
  3.2.5.3 Transparent content models ........................................................................................................ 134
  3.2.5.4 Paragraphs .................................................................................................................................... 135
3.2.6 Global attributes ............................................................................................................................... 137
  3.2.6.1 The title attribute ...................................................................................................................... 140
  3.2.6.2 The lang and xml:lang attributes .............................................................................................. 141
  3.2.6.3 The translate attribute ................................................................................................................. 142
  3.2.6.4 The dir attribute .......................................................................................................................... 143
  3.2.6.5 The style attribute ....................................................................................................................... 145
  3.2.6.6 Embedding custom non-visible data with the data-* attributes ............................................ 146
3.2.7 The innerText getter and setter ..................................................................................................... 149
3.2.8 Requirements relating to the bidirectional algorithm .................................................................... 151
  3.2.8.1 Authoring conformance criteria for bidirectional-algorithm formatting characters .................... 151
  3.2.8.2 User agent conformance criteria .................................................................................................. 151
3.2.9 Requirements related to ARIA and to platform accessibility APIs .................................................... 152

4 The elements of HTML .......................................................................................................................... 153
  4.1 The document element ...................................................................................................................... 153
    4.1.1 The html element ........................................................................................................................ 153
  4.2 Document metadata .......................................................................................................................... 154
    4.2.1 The head element ........................................................................................................................ 154
    4.2.2 The title element .......................................................................................................................... 155
    4.2.3 The base element ......................................................................................................................... 156
    4.2.4 The link element .......................................................................................................................... 158
      4.2.4.1 Processing the media attribute ............................................................................................. 162
      4.2.4.2 Processing the type attribute ............................................................................................... 163
      4.2.4.3 Fetching and processing a resource from a link element ..................................................... 164
      4.2.4.4 Processing 'Link' headers ..................................................................................................... 165
      4.2.4.5 Providing users with a means to follow hyperlinks created using the link element .............. 165
    4.2.5 The meta element ......................................................................................................................... 165
      4.2.5.1 Standard metadata names ..................................................................................................... 166
      4.2.5.2 Other metadata names ........................................................................................................... 170
      4.2.5.3 Pragma directives .................................................................................................................... 171
      4.2.5.4 Specifying the document's character encoding .................................................................... 175
    4.2.6 The style element ......................................................................................................................... 176
    4.2.7 Interactions of styling and scripting ............................................................................................. 178
  4.3 Sections ................................................................................................................................................. 179
    4.3.1 The body element ....................................................................................................................... 179
4.3.2 The article element .......................................................... 181
4.3.3 The section element ........................................................ 183
4.3.4 The nav element .............................................................. 186
4.3.5 The aside element ........................................................... 189
4.3.6 The h1, h2, h3, h4, h5, and h6 elements ......................... 191
4.3.7 The hgroup element ......................................................... 193
4.3.8 The header element ........................................................ 195
4.3.9 The footer element .......................................................... 196
4.3.10 The address element ...................................................... 199
4.3.11 Headings and sections .................................................... 200
  4.3.11.1 Creating an outline .................................................. 202
  4.3.11.2 Sample outlines ...................................................... 206
  4.3.11.3 Exposing outlines to users ........................................ 211
4.3.12 Usage summary ............................................................ 211
  4.3.12.1 Article or section? .................................................... 213
4.4 Grouping content ................................................................ 213
  4.4.1 The p element ............................................................... 213
  4.4.2 The hr element ............................................................. 215
  4.4.3 The pre element ........................................................... 217
  4.4.4 The blockquote element ................................................ 219
  4.4.5 The ol element ............................................................. 222
  4.4.6 The ul element ............................................................. 223
  4.4.7 The menu element ....................................................... 224
  4.4.8 The li element ............................................................. 225
  4.4.9 The dl element ............................................................ 228
  4.4.10 The dt element ........................................................... 231
  4.4.11 The dd element ........................................................... 232
  4.4.12 The figure element ...................................................... 233
  4.4.13 The figcaption element ............................................... 236
  4.4.14 The main element ....................................................... 237
  4.4.15 The div element ........................................................ 238
4.5 Text-level semantics ............................................................ 239
  4.5.1 The a element ............................................................... 239
  4.5.2 The em element ............................................................ 242
  4.5.3 The strong element ....................................................... 243
  4.5.4 The small element ........................................................ 245
  4.5.5 The s element ............................................................... 246
  4.5.6 The cite element .......................................................... 247
  4.5.7 The q element .............................................................. 248
  4.5.8 The dfn element ........................................................... 250
  4.5.9 The abbr element ........................................................ 251
  4.5.10 The ruby element ......................................................... 253
  4.5.11 The rt element ............................................................ 259
  4.5.12 The rp element ............................................................ 259
  4.5.13 The data element ........................................................ 260
  4.5.14 The time element ........................................................ 261
  4.5.15 The code element ........................................................ 268
  4.5.16 The var element ........................................................ 269
  4.5.17 The samp element ....................................................... 270
  4.5.18 The kbd element ........................................................ 271
4.8 Embedded content

4.8.1 The picture element

4.8.2 The source element

4.8.3 The img element

4.8.4 Images

4.8.4.1 Introduction

4.8.4.2 Attributes common to source, img, and link elements

4.8.4.3 Processing model

4.8.4.4 Requirements for providing text to act as an alternative for images

4.8.5 The iframe element

4.8.6 The embed element

4.8.7 The object element

4.8.8 The param element

4.8.9 The video element

4.8.10 The audio element

4.8.11 The track element

4.8.12 Media elements

4.8.12.1 Error codes

4.8.12.2 Location of the media resource

4.8.12.3 MIME types

4.8.12.4 Network states

4.8.12.5 Loading the media resource
The Techniques for table design

Security and privacy considerations

Best practices for metadata text tracks

The Selecting specific audio and video tracks declaratively

The Best practices for authors using media elements

Writing a form's user interface

Identifying a track kind through a URL

The Implementing the server-side processing for a form

4.1 The map element

4.14 The area element

4.15 Image maps

4.15.1 Authoring

4.15.2 Processing model

4.16 MathML

4.17 SVG

4.18 Dimension attributes

4.9 Tabular data

4.9.1 The table element

4.9.1.1 Techniques for describing tables

4.9.1.2 Techniques for table design

4.9.2 The caption element

4.9.3 The colgroup element

4.9.4 The col element

4.9.5 The tbody element

4.9.6 The thead element

4.9.7 Thetfoot element

4.9.8 The tr element

4.9.9 The td element

4.9.10 The th element

4.9.11 Attributes common to td and th 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
6.5.7 The autofocus attribute

6.6 Assigning keyboard shortcuts

6.6.1 Introduction

6.6.2 The accesskey attribute

6.6.3 Processing model

6.7 Editing

6.7.1 Making document regions editable: The contenteditable content attribute

6.7.2 Making entire documents editable: the designMode getter and setter

6.7.3 Best practices for in-page editors

6.7.4 Editing APIs

6.7.5 Spelling and grammar checking

6.7.6 Autocapitalization

6.7.7 Input modalities: the inputmode attribute

6.7.8 Input modalities: the enterkeyhint attribute

6.8 Find-in-page

6.9 Drag and drop

6.9.1 Introduction

6.9.2 The drag data store

6.9.3 The DataTransfer interface

6.9.4 The DragEvent interface

7 Loading web pages

7.1 Browsing contexts

7.1.1 Creating browsing contexts

7.1.2 Related browsing contexts

7.1.2.1 Navigating related browsing contexts in the DOM

7.1.3 Security

7.1.4 Groupings of browsing contexts

7.1.5 Browsing context names

7.2 Security infrastructure for Window, WindowProxy, and Location objects

7.2.1 Integration with IDL

7.2.2 Shared internal slot: [[CrossOriginPropertyDescriptorMap]]

7.2.3 Shared abstract operations

7.2.3.1 CrossOriginProperties (O)

7.2.3.2 CrossOriginPropertyFallback (P)

7.2.3.3 IsPlatformObjectSameOrigin (O)

7.2.3.4 CrossOriginGetOwnPropertyHelper (O, Receiver)

7.2.3.5 CrossOriginGet (O, P, Receiver)

7.2.3.6 CrossOriginSet (O, P, V, Receiver)

7.2.3.7 CrossOriginOwnPropertyKeys (O)

7.3 The Window object

7.3.1 APIs for creating and navigating browsing contexts by name

7.3.2 Accessing other browsing contexts
8 Web application APIs .......................................................................................................................................................................................... 899

 8.1 Scripting .................................................................................................................................................................................................................... 899

 8.1.1 Introduction .................................................................................................................................................................................................. 899

 8.1.2 Agents and agent clusters ........................................................................................................................................................................... 899

 8.1.2.1 Integration with the JavaScript agent formalism ................................................................................................................................. 899

 8.1.2.2 Integration with the JavaScript agent cluster formalism .................................................................................................................. 900

 8.1.3 Realms and their counterparts ........................................................................................................................................................................ 902

 8.1.3.1 Environments .................................................................................................................................................................................................. 902

 8.1.3.2 Environment settings objects ................................................................................................................................................................ 903

 8.1.3.3 Realms, settings objects, and global objects ............................................................................................................................................... 904

 8.1.3.3.1 Entry .................................................................................................................................................................................................. 907

 8.1.3.3.2 Incumbent .......................................................................................................................................................................................... 907

 8.1.3.3.3 Current .................................................................................................................................................................................................. 910

 8.1.3.3.4 Relevant .................................................................................................................................................................................................. 910

 8.1.3.4 Enabling and disabling scripting ....................................................................................................................................................... 910

 8.1.3.5 Secure contexts .................................................................................................................................................................................................. 910

 8.1.4 Script processing model ............................................................................................................................................................................ 911

 8.1.4.1 Scripts .................................................................................................................................................................................................. 911

 8.1.4.2 Fetching scripts .................................................................................................................................................................................................. 912

 8.1.4.3 Creating scripts .................................................................................................................................................................................................. 919

 8.1.4.4 Calling scripts .................................................................................................................................................................................................. 920

 8.1.4.5 Killing scripts .................................................................................................................................................................................................. 922

 8.1.4.6 Runtime script errors ............................................................................................................................................................................ 923

 8.1.4.7 Unhandled promise rejections ....................................................................................................................................................... 924

 8.1.5 JavaScript specification host hooks ............................................................................................................................................................. 925

 8.1.5.1 HostEnsureCanCompileStrings(callerRealm, calleeRealm) .................................................................................................................. 925

 8.1.5.2 HostPromiseRejectionTracker(promise_operation) ...................................................................................................................... 925

 8.1.5.3 Job-related host hooks ........................................................................................................................................................................ 926

 8.1.5.3.1 HostCallJobCallback(callback, V, argumentsList) ...................................................................................................................... 926

 8.1.5.3.2 HostEnqueuePromiseJob(job, realm) ................................................................................................................................................... 926

 8.1.5.3.3 HostMakeJobCallback(caller) .......................................................................................................................................................... 927

 8.1.5.4 Module-related host hooks ...................................................................................................................................................................... 928

 8.1.5.4.1 HostGetImportMetaProperties(moduleRecord) ........................................................................................................................................ 929

 8.1.5.4.2 HostImportModuleDynamically(referencingScriptOrModule, specifier, promiseCapability) ........................................................................ 929

 8.1.5.4.3 HostResolveImportedModule(referencingScriptOrModule, specifier) .............................................................................................. 930

 8.1.6 Event loops .................................................................................................................................................................................................. 931

 8.1.6.1 Definitions .................................................................................................................................................................................................. 931

 8.1.6.2 Queuing tasks .................................................................................................................................................................................................. 932
13.2.4 Parse state ........................................................................................................... 1084
  13.2.4.1 The insertion mode ......................................................................................... 1084
  13.2.4.2 The stack of open elements .......................................................................... 1085
  13.2.4.3 The list of active formatting elements ......................................................... 1087
  13.2.4.4 The element pointers ..................................................................................... 1088
  13.2.4.5 Other parsing state flags ................................................................................ 1088
13.2.5 Tokenization ......................................................................................................... 1089
  13.2.5.1 Data state ......................................................................................................... 1090
  13.2.5.2 RAWTEXT state ............................................................................................ 1090
  13.2.5.3 RAWTEXT state ............................................................................................ 1090
  13.2.5.4 Script data state ........................................................................................... 1090
  13.2.5.5 PLAINTEXT state .......................................................................................... 1091
  13.2.5.6 Tag open state ............................................................................................... 1091
  13.2.5.7 End tag open state ........................................................................................ 1091
  13.2.5.8 Tag name state .............................................................................................. 1092
  13.2.5.9 RC DATA less-than sign state ....................................................................... 1092
  13.2.5.10 RC DATA end tag open state ...................................................................... 1092
  13.2.5.11 RC DATA end tag name state ..................................................................... 1093
  13.2.5.12 RAWTEXT less-than sign state .................................................................. 1093
  13.2.5.13 RAWTEXT end tag open state .................................................................... 1093
  13.2.5.14 RAWTEXT end tag name state ................................................................... 1093
  13.2.5.15 Script data less-than sign state .................................................................. 1094
  13.2.5.16 Script data end tag open state .................................................................... 1094
  13.2.5.17 Script data end tag name state ................................................................... 1094
  13.2.5.18 Script data escape start state ...................................................................... 1095
  13.2.5.19 Script data escape start dash state ............................................................. 1095
  13.2.5.20 Script data escaped state ........................................................................... 1095
  13.2.5.21 Script data escaped dash dash state ........................................................... 1096
  13.2.5.22 Script data escaped dash dash dash state .................................................. 1096
  13.2.5.23 Script data escaped less-than sign state .................................................... 1096
  13.2.5.24 Script data escaped end tag open state ...................................................... 1097
  13.2.5.25 Script data escaped end tag name state .................................................... 1097
  13.2.5.26 Script data double escape start state ......................................................... 1097
  13.2.5.27 Script data double escaped state .............................................................. 1098
  13.2.5.28 Script data double escaped dash dash state .............................................. 1098
  13.2.5.29 Script data double escaped dash dash dash state ...................................... 1099
  13.2.5.30 Script data double escaped less-than sign state ....................................... 1099
  13.2.5.31 Script data double escape end state .......................................................... 1099
  13.2.5.32 Before attribute name state ......................................................................... 1100
  13.2.5.33 Attribute name state ................................................................................... 1100
  13.2.5.34 After attribute name state ........................................................................... 1101
  13.2.5.35 Before attribute value state ......................................................................... 1101
  13.2.5.36 Attribute value (double-quoted) state ......................................................... 1101
  13.2.5.37 Attribute value (single-quoted) state .......................................................... 1102
  13.2.5.38 Attribute value (unquoted) state ................................................................. 1102
  13.2.5.39 After attribute value (quoted) state ............................................................ 1103
  13.2.5.40 Self-closing start tag state .......................................................................... 1103
  13.2.5.41 Bogus comment state .................................................................................. 1103
  13.2.5.42 Markup declaration open state ................................................................... 1104
  13.2.5.43 Comment start state ................................................................................... 1104
15.5.2 Button layout.................................................................................................................. 1204
15.5.3 The button element....................................................................................................... 1205
15.5.4 The details and summary elements ........................................................................... 1205
15.5.5 The input element as a text entry widget ................................................................. 1205
15.5.6 The input element as domain-specific widgets ....................................................... 1206
15.5.7 The input element as a range control ....................................................................... 1206
15.5.8 The input element as a color well ............................................................................. 1206
15.5.9 The input element as a checkbox and radio button widgets .................................... 1206
15.5.10 The input element as a file upload control ............................................................ 1207
15.5.11 The input element as a button ................................................................................... 1207
15.5.12 The marquee element .............................................................................................. 1207
15.5.13 The meter element .................................................................................................... 1208
15.5.14 The progress element .............................................................................................. 1208
15.5.15 The select element .................................................................................................... 1209
15.5.16 The textarea element ............................................................................................. 1210
15.6 Frames and frame sets ..................................................................................................... 1210
15.7 Interactive media ............................................................................................................. 1212
15.7.1 Links, forms, and navigation ...................................................................................... 1212
15.7.2 The title attribute ....................................................................................................... 1212
15.7.3 Editing hosts ................................................................................................................ 1213
15.7.4 Text rendered in native user interfaces ..................................................................... 1213
15.8 Print media ...................................................................................................................... 1215
15.9 Unstyled XML documents .............................................................................................. 1215

16 Obsolete features ................................................................................................................ 1216
16.1 Obsolete but conforming features ................................................................................ 1216
16.1.1 Warnings for obsolete but conforming features ....................................................... 1216
16.2 Non-conforming features .............................................................................................. 1217
16.3 Requirements for implementations .............................................................................. 1222
16.3.1 The marquee element .............................................................................................. 1222
16.3.2 Frames ....................................................................................................................... 1224
16.3.3 Other elements, attributes and APIs ......................................................................... 1225

17 IANA considerations .......................................................................................................... 1236
17.1 text/html ......................................................................................................................... 1236
17.2 multipart/x-mixed-replace ............................................................................................ 1237
17.3 application/xhtml+xml ................................................................................................. 1238
17.4 text/ping ........................................................................................................................ 1239
17.5 application/microdata+json .......................................................................................... 1240
17.6 text/event-stream .......................................................................................................... 1241
17.7 'Cross-Origin-Embedder-Policy' ................................................................................... 1242
17.8 'Cross-Origin-Embedder-Policy-Report-Only' ............................................................ 1242
17.9 'Cross-Origin-Opener-Policy' ....................................................................................... 1242
17.10 'Cross-Origin-Opener-Policy-Report-Only' ............................................................... 1243
17.11 'Origin-Agent-Cluster' ............................................................................................... 1243
17.12 'Ping-From' ................................................................................................................ 1244
17.13 'Ping-To' ..................................................................................................................... 1244
17.14 'Refresh' ...................................................................................................................... 1244
This specification defines a big part of the web platform, in lots of detail. Its place in the web platform specification stack relative to other specifications can be best summed up as follows:

<table>
<thead>
<tr>
<th>CSS</th>
<th>SVG</th>
<th>MathML</th>
<th>Service Workers</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSP</td>
<td>PNG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fetch</td>
<td>Opus</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AV1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

HTTP | TLS | DOM | Unicode | Web IDL | MIME | URL | XML | JavaScript | Encoding |

### 1.2 Is this HTML5?  
This section is non-normative.

In short: Yes.
In more length: the term "HTML5" is widely used as a buzzword to refer to modern web technologies, many of which (though by no means all) are developed at the WHATWG. This document is one such; others are available from the WHATWG Standards overview.

1.3 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.

1.4 Audience

This section is non-normative.

This specification is intended for authors of documents and scripts that use the features defined in this specification, implementers 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.

1.5 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), public telephone books or address books, communications software (email clients, instant messaging clients, discussion software), document editing software, etc.

1.6 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. HTML4 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 HTML4 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 XHTML2.

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 HTML4’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 HTML5 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 HTML5 specification include what had previously been specified in three separate documents: HTML4, XHTML1, and DOM2 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 HTML5 after all, and in 2007 formed a working group chartered to work with the WHATWG on the development of the HTML5 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. In 2011, however, the groups came to the conclusion that they had different goals: the W3C wanted to publish a “finished” version of “HTML5", while the WHATWG wanted to continue working on a Living Standard for HTML, continuously maintaining the specification rather than freezing it in a state with known problems, and adding new features as needed to evolve the platform.

In 2019, the WHATWG and W3C signed an agreement to collaborate on a single version of HTML going forward: this document.

1.7 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.

1.7.1 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.

The exception to this general design principle is the JavaScript SharedArrayBuffer class. Using SharedArrayBuffer objects, it can in fact be observed that scripts in other agents are executing simultaneously. Furthermore, due to the JavaScript memory model, there are situations which not only are un-representable via serialized script execution, but also un-representable via serialized statement execution among those scripts.

1.7.2 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.

1.7.3 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 mechanism to include page-wide metadata.

- 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 mechanism with a custom type, for further handling by inline or server-side scripts.

- Authors can create plugins and invoke them using the element. This is how Flash works.

- Authors can extend APIs using the JavaScript prototyping mechanism. This is widely used by script libraries, for instance.

- Authors can use the microdata feature (the itemprop attributes) to embed nested name-value pairs of data to be shared with other applications and sites.
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 HTML syntax, known simply as "HTML".

The second concrete syntax is 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.

The DOM, the HTML syntax, and the XML 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 XML 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 XML syntax. Comments that contain the string "--->" can only be represented in the DOM, not in the HTML and XML syntaxes.

This specification is divided into the following major sections:

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

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

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.

The elements of HTML
Each element has a predefined meaning, which is explained in this section. Rules for authors on how to use the element, 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.

Microdata
This specification introduces a mechanism for adding machine-readable annotations to documents, so that tools can extract trees of name-value pairs from the document. This section describes this mechanism and some algorithms that can be used to convert HTML documents into other formats. This section also defines some sample Microdata vocabularies for contact information, calendar events, and licensing works.

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.
**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.

**Web application APIs**

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

**Web workers**

This section defines an API for background threads in JavaScript.

**The communication APIs**

This section describes some mechanisms that applications written in HTML can use to communicate with other applications from different domains running on the same client. It also introduces a server-push event stream mechanism known as Server Sent Events or **EventSource**

**Web storage**

This section defines a client-side storage mechanism based on name-value pairs.

**The HTML syntax**

**The XML 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 XML, along with rules for how to parse content using those syntaxes.

**Rendering**

This section defines the default rendering rules for web browsers.

There are also some appendices, listing **obsolete features** and **IANA considerations**, and several indices.

### 1.9.1 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**

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**

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.

### 1.9.2 Typographic conventions

This is a definition, requirement, or explanation.
This is an open issue.

⚠ Warning!
This is a warning.

```
[Exposed=Window]
interface Example {
  // this is an IDL definition
};
```

For web developers (non-normative)

```
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.

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.

1.10 A quick introduction to HTML

This section is non-normative.

A basic HTML document looks like this:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Sample page</title>
  </head>
  
  <!-- body content goes here -->
</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:

```html
<p>This is <em>very</em> wrong!</p>
<p>This <em>is</em> correct.</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:

```html
<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 ASCII whitespace or any of "'" ` `<` >. 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.

```html
<!-- empty attributes -->
<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:

```
DOCTYPE: html
  html lang="en"
    head
      title
        #text: Sample page
      #text: Sample page
    body
      p
        #text: This is a
```
The document element of this tree is the html element, which is the element always found in that position in 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 lang="en">
<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>
```
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.

1.10.1 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][1276]

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

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:

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

```html
https://example.com/message.cgi?say=%3Cscript%3Ealert%28%27Oh%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>`, it is important to safelist any provided attributes as well. 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).

1.10.2 Common pitfalls to avoid when using the scripting APIs

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 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.

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
</table>

One way this could manifest itself is with img↵ 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 img↵ element to catch the load↵ event:

```html
<img src="games.png" alt="Games" onload="gamesLogoHasLoaded(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>
This section is non-normative.

Authors are encouraged to make use of conformance checkers (also known as validators) to catch common mistakes. The WHATWG maintains a list of such tools at: https://whatwg.org/validator/

1.10.3 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 WHATWG maintains a list of such tools at: https://whatwg.org/validator/

1.11 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.

1.11.1 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, even using such techniques doesn't help make pages accessible for non-AT non-graphical users, such as users of 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 work 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; HTML4 deprecated presentational markup many years ago and provided a mode (HTML4 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 in syndication 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: `b`, `i`, `hr`, `s`, `small`, and `u`.

### 1.11.2 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**

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 element or attribute names never contain multiple colons, 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. (Comments containing two consecutive hyphens, or ending with a hyphen, are exceptions that are allowed in the HTML syntax.)

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**

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

```
<p><i>She dreamt.</i>
<p><i>She dreamt that she ate breakfast.</i>
<p><i>Then lunch.</i>
<p><i>And finally dinner.</i>
```
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

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 "&copy;" 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&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

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.
Errors that risk exposing authors to security attacks

Certain restrictions exist purely to avoid known security problems.

Example

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. \[\text{UTF7}\]

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

For example, it is unclear whether the author intended the following to be an \texttt{h1}\hyperref[191]{191} heading or an \texttt{h2}\hyperref[191]{191} heading:

\[
\text{<h1>\text{Contact details</h2>}}
\]

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 debugging 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

For example, if the author typed \texttt{<capton>} instead of \texttt{<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

For example, “attributes” in end tags are ignored currently, but they are invalid, in case a future change to the language makes use of that syntax feature 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.

1.11.3 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

For example, this specification disallows nesting a \texttt{section}\hyperref[183]{183} element inside a \texttt{kbd}\hyperref[271]{271} 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

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.

```html
<hr role="cell">
<input type=radio role=progressbar>
```

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

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.

Errors that indicate a likely misunderstanding of the specification

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

Example

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

For example, the `area` element's `shape` attribute, despite accepting both `circle` 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

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:
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
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
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 between the HTML and XML syntaxes
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 the HTML and XML syntaxes.

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

Example
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
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
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.
1.12 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]^{p1270}**

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 [UTR36]^{p1277}**

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) [WCAG]^{p1277}**

Web Content Accessibility Guidelines (WCAG) 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 [ATAG]^{p1270}**

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 [UAAG]^{p1277}**

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.
2 Common infrastructure  

This specification depends on Infra. [INFRA]p1273

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 **XML 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 XML)".

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 the **XML syntax**, depending on context.

In the context of the DOM structures, the terms **HTML document** and **XML document** are used as defined in **DOM**, and refer specifically to two different modes that **Document** objects can find themselves in. [DOM]p1272 (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**.

For simplicity, terms such as **shown**, **displayed**, and **visible** might sometimes be used 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.

2.1.1 Parallelism  

To run steps **in parallel** means those steps are to be run, one after another, at the same time as other logic in the standard (e.g., at the same time as the **event loop**). This standard 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.

**Note**

For guidance on writing specifications that leverage parallelism, see **Dealing with the event loop from other specifications**.

To avoid race conditions between different **in parallel** algorithms that operate on the same data, a **parallel queue** can be used.

A **parallel queue** represents a queue of algorithm steps that must be run in series.

A **parallel queue** has an **algorithm queue** (a **queue**), initially empty.

To **enqueue steps** to a **parallel queue**, **enqueue** the algorithm steps to the **parallel queue**'s **algorithm queue**.

To **start a new parallel queue**, run the following steps:

1. Let **parallelQueue** be a new **parallel queue**.
2. Run the following steps **in parallel**:
1. While true:
   1. Let steps be the result of `dequeueing` from `parallelQueue`'s `algorithm queue`\(^{342}\).
   2. If steps is not nothing, then run steps.
   3. Assert: running steps did not throw an exception, as steps running in parallel\(^{342}\) are not allowed to throw.

   **Note**
   Implementations are not expected to implement this as a continuously running loop. Algorithms in standards are to be easy to understand and are not necessarily great for battery life or performance.

3. Return `parallelQueue`.

   **Note**
   Steps running in parallel\(^{342}\) can themselves run other steps in in parallel\(^{342}\). E.g., inside a `parallel queue`\(^{342}\) it can be useful to run a series of steps in parallel with the queue.

**Example**

Imagine a standard defined `nameList` (a list), along with a method to add a `name` to `nameList`, unless `nameList` already contains `name`, in which case it rejects.

The following solution suffers from race conditions:

1. Let p be a new promise.
2. Run the following steps in parallel\(^{342}\):
   1. If `nameList contains` name, reject p with a `TypeError` and abort these steps.
   2. Do some potentially lengthy work.
   3. Append `name` to `nameList`.
   4. Resolve p with undefined.

Two invocations of the above could run simultaneously, meaning `name` isn't in `nameList` during step 2.1, but it might be added before step 2.3 runs, meaning `name` ends up in `nameList` twice.

Parallel queues solve this. The standard would let `nameListQueue` be the result of starting a new parallel queue\(^{342}\), then:

1. Let p be a new promise.
2. Enqueue the following steps\(^{342}\) to `nameListQueue`:
   1. If `nameList contains` name, reject p with a `TypeError` and abort these steps.
   2. Do some potentially lengthy work.
   3. Append `name` to `nameList`.
   4. Resolve p with undefined.

The steps would now queue and the race is avoided.

### 2.1.2 Resources\(^{343}\)

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

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

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 specifications, refer to as a representation is referred to in this specification as a resource. [HTTP][p1273]

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.

For CSS style sheets, we tentatively define here that their critical subresources are other style sheets imported via @import rules, including those indirectly imported by other imported style sheets.

This definition is not fully interoperable; furthermore, some user agents seem to count resources like background images or web fonts as critical subresources. Ideally, the CSS Working Group would define this; see w3c/csswg-drafts issue #1088 to track progress on that front.

2.1.3 XML compatibility [p44]

To ease migration from HTML to XML, UAs 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” refers to any element in that namespace, even in XML documents.

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 (:).

2.1.4 DOM trees [p44]

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.

The term empty, when used for an attribute value, Text node, or string, means that the length of the text is zero (i.e., not even containing controls or U+0020 SPACE).

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.

A node is inserted into a document when the insertion steps are invoked with it as the argument and it is now in a document tree. Analogously, a node is removed from a document when the removing steps are invoked with it as the argument and it is now no longer in a document tree.
A node **becomes connected** when the insertion steps are invoked with it as the argument and it is now connected. Analogously, a node **becomes disconnected** when the removing steps are invoked with it as the argument and it is now no longer connected.

A node is **browsing-context connected** when it is connected and its shadow-including root’s browsing context is non-null. A node becomes **browsing-context connected** when the insertion steps are invoked with it as the argument and it is now browsing-context connected. A node becomes **browsing-context disconnected** either when the removing steps are invoked with it as the argument and it is now no longer browsing-context connected, or when its shadow-including root’s browsing context becomes null.

### 2.1.5 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.

### 2.1.6 Plugins

The term **plugin** refers to an implementation-defined set of content handlers used by the user agent that 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**

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 UAs 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**

For example, a secured plugin would prevent its contents from creating popups 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 those in the user agent.
Since different users having different sets of plugins provides a tracking vector that increases the chances of users being uniquely identified, user agents are encouraged to support the exact same set of plugins for each user.

### 2.1.7 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 Encoding. 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]

### 2.1.8 Conformance classes

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

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**

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 XML 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**

A conforming web browser would, upon finding a 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 Web IDL. [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 XML 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.
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 the rendering section defines as 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.

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.

Conformant 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 XML 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
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
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 its nearest article or body element ancestor. 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 that when the user uses a tool for inserting contact information for an article element, that the user really is doing that and not inserting something else instead.

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, 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.)
This specification relies on several other underlying specifications.

**Infra**

The following terms are defined in *Infra*: [INFRA](#)

- The general iteration terms *while, continue*, and *break*.
- Implementation-defined
- Tracking vector
- Code point and its synonym character
- Surrogate
- Scalar value
- Tuple
- Noncharacter
- String, code unit, length, and code point length
- The string equality operations *is* and *identical to*
- Scalar value string
- Convert
- ASCII whitespace
- Control
- ASCII digit
- ASCII upper hex digit
- ASCII lower hex digit
- ASCII hex digit
- ASCII upper alpha
- ASCII lower alpha
- ASCII alpha
- ASCII alphanumeric
- Isomorphic decode
- ASCII lowercase
- ASCII uppercase
- ASCII case-insensitive
- Strip newlines
- Normalize newlines
- Strip leading and trailing ASCII whitespace
- Strip and collapse ASCII whitespace
- Split a string on ASCII whitespace
- Split a string on commas
- Collect a sequence of code points and its associated position variable
- Skip ASCII whitespace
- The ordered map data structure and the associated definitions for value, entry, exists, getting the value of an entry, setting the value of an entry, removing an entry, clear, getting the keys, size, and iterate.
- The list data structure and the associated definitions for append, extend, replace, remove, empty, contains, size, is empty, iterate, and clone.
- The stack data structure and the associated definitions for push and pop.
- The queue data structure and the associated definitions for enqueue and dequeue.
- The ordered set data structure and the associated definition for append and union.
- The struct specification type and the associated definition for item.
- The forgiving-base64 encode and forgiving-base64 decode algorithms.
- HTML namespace
- MathML namespace
- SVG namespace
- XLink namespace
- XML namespace
- XMLNS namespace

**Unicode and Encoding**

The Unicode character set is used to represent textual data, and *Encoding* 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 *Encoding*: [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 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
- The BOM sniff algorithm which takes a byte stream and returns an encoding or null.
XML and related specifications

Implementations that support [XML][p1179] for HTML must support some version of XML, as well as its corresponding namespaces specification, because that syntax uses an XML serialization with namespaces. [XML][p1278] [XMLNS][p1278]

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 analogues are in certain namespaces, without actually exposing the namespace strings.

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.

The attribute with the name space in the XML namespace is defined by Extensible Markup Language (XML). [XML][p1278]

The Name production is defined in XML. [XML][p1278]

This specification also references the <?xml-stylesheet?> processing instruction, defined in Associating Style Sheets with XML documents. [XMLSSPI][p1278]

This specification also non-normatively mentions the XSLTProcessor interface and its transformToFragment() and transformToDocument() methods. [XSLTP][p1278]

URLs

The following terms are defined in URL: [URL][p1277]

- host
- public suffix
- domain
- IPv4 address
- IPv6 address
- URL
- Origin of URLs
- Absolute URL
- Relative URL
- registrable domain
- The URL parser
  - The basic URL parser and its url and state override arguments, 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
    - cannot-be-a-base-URL
    - blob URL entry
- valid URL string
- The cannot have a username/password/port concept
- URL serializer and its exclude fragment argument
- The host parser
- The host serializer
- Host equals
- URL equals and its exclude fragments argument
- serialize an integer
- Default encode set
- component percent-encode set
- UTF-8 percent-encode
- percent-decode
- set the username
- set the password
- The application/x-www-form-urlencoded format
- The application/x-www-form-urlencoded serializer

A number of schemes and protocols are referenced by this specification also:
The following terms are defined in the HTTP specifications: [HTTP][p1273]

- `Accept` header
- `Accept-Language` header
- `Cache-Control` header
- `Content-Disposition` header
- `Content-Language` header
- `Last-Modified` header
- `Referer` header

The following terms are defined in HTTP State Management Mechanism: [COOKIES][p1270]

- cookie-string
- receives a set-cookie-string
- `Cookie` header

The following term is defined in Web Linking: [WEBLINK][p1277]

- `Link` header

The following terms are defined in Structured Field Values for HTTP: [STRUCTURED-FIELDS][p1276]

- structured header
- boolean
- token
- parameters

The following terms are defined in MIME Sniffing: [MIMESNIFF][p1274]

- MIME type
- valid MIME type string
- valid MIME type string with no parameters
- HTML MIME type
- JavaScript MIME type and JavaScript MIME type essence match
- JSON MIME type
- XML MIME type

Fetch

The following terms are defined in Fetch: [FETCH][p1272]

- ABNF
- about:blank
- An HTTP(S) scheme
- A local scheme
- A fetch scheme
- CORS protocol
default ‘User-Agent’ value
- extract a MIME type
- fetch
- HTTP-redirect fetch
- ok status
- navigation request
- network error
- `Origin` header
- `Cross-Origin-Resource-Policy` header
- process response
- getting a structured field value
- set
get, decode, and split
- terminate
cross-origin resource policy check
- the RequestCredentials enumeration
- the RequestDestination enumeration
- the fetch() method
- serialize a response URL for reporting
- response and its associated:
The following terms are defined in Referrer Policy: [REFERRERPOLICY]

- referer policy
- The `Referrer-Policy` HTTP header
- The parse a referer policy from a `Referrer-Policy` header algorithm
- The “no-referrer”, “no-referrer-when-downgrade”, “origin-when-cross-origin”, and “unsafe-url” referer policies
- The default referer policy

The following terms are defined in Mixed Content: [MIX]

- a priori authenticated URL

Paint Timing

The following terms are defined in Paint Timing: [PAINTTIMING]

- mark paint timing

Long Tasks

The following terms are defined in Long Tasks: [LONGTASKS]

- report long tasks

Web IDL

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

The following terms are defined in Web IDL:

- this
- extended attribute
- named constructor
- constructor operation
- overridden constructor steps
- internally create a new object implementing the interface
- array index property name
- supported property indices
- determine the value of an indexed property
- set the value of an existing indexed property
set the value of a new indexed property
support named properties
supported property names
determine the value of a named property
set the value of an existing named property
set the value of a new named property
delete an existing named property
perform a security check
platform object
legacy platform object
primary interface
interface object
include
inherit
interface prototype object
[(Realm)] field of a platform object
callback context
frozen array and creating a frozen array
create a new object implementing the interface
callback this value
converting between Web IDL types and JS types
invoking and constructing callback functions
overload resolution algorithm
exposed
a promise rejected with
upon rejection
[LegacyFactoryFunction]
[LegacyLenientThis]
[LegacyNullToString]
[LegacyOverrideBuiltIns]
[LegacyTreatNonObjectAsNull]
[LegacyUnenumerableNamedProperties]
[LegacyUnforgeable]
The Web IDL also defines the following types that are used in Web IDL fragments in this specification:

- ArrayBuffer
- ArrayBufferView
- boolean
- DOMString
- double
- enumeration
- Function
- long
- object
- Uint8ClampedArray
- unrestricted double
- unsigned long
- USVString
- VoidFunction

The term `throw` in this specification is used as defined in Web IDL. The `DOMException` type and the following exception names are defined by Web IDL and used by this specification:

- "IndexSizeError"
- "HierarchyRequestError"
- "InvalidCharacterError"
- "NotFoundError"
- "NotSupportedError"
- "InvalidStateError"
- "SyntaxError"
- "NetworkError"
- "AbortError"
- "QuotaExceededError"
- "DataCloneError"
- "EncodingError"
- "NotAllowedError"

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
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
JavaScript

Some parts of the language described by this specification only support JavaScript as the underlying scripting language.

Note

The term "JavaScript" is used to refer to ECMA-262, 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\[RFC4329\] according to RFC 4329. [RFC4329]\[RFC4329\]

The following terms are defined in the JavaScript specification and used in this specification:

- active function object
- agent and agent cluster
- automatic semicolon insertion
- candidate execution
- The current Realm Record
- early error
- forward progress
- invariants of the essential internal methods
- JavaScript execution context
- JavaScript execution context stack
- JavaScript realm
- JobCallback Record
- EnvironmentRecord
- NewTarget
- running JavaScript execution context
- surrounding agent
- abstract closure
- immutable prototype exotic object
- Well-Known Symbols, including @@hasInstance, @@isConcatSpreadable, @@toPrimitive, and @@toStringTag
- Well-Known Intrinsic Objects, including %Array.prototype%, %Error.prototype%, %EvalError.prototype%, %Function.prototype%, %JSON.parse%, %Object.prototype%, %Object.prototype.valueOf%, %RangeError.prototype%, %ReferenceError.prototype%, %SyntaxError.prototype%, %TypeError.prototype%, and %URIError.prototype%
- The FunctionBody production
- The Module production
- The Pattern production
- The Script production
- The Type notation
- The Completion Record specification type
- The List and Record specification types
- The Property Descriptor specification type
- The Script Record specification type
- The Cyclic Module Record specification type
- The Source Text Module Record specification type and its Evaluate and Link methods
- The ArrayCreate abstract operation
- The Call abstract operation
- The Construct abstract operation
- The CopyDataBlockBytes abstract operation
- The CreateByteDataBlock abstract operation
- The CreateDataProperty abstract operation
- The DetachArrayBuffer abstract operation
- The EnumerableOwnPropertyNames abstract operation
- The FinishDynamicImport abstract operation
- The OrdinaryFunctionCreate abstract operation
- The Get abstract operation
- The GetActiveScriptOrModule abstract operation
- The GetFunctionRealm abstract operation
- The HasOwnProperty abstract operation
- The HostCallJobCallback abstract operation
- The HostEnqueuePromiseJob abstract operation
- The HostEnsureCanCompileStrings abstract operation
- The HostImportModuleDynamically abstract operation
- The HostMakeJobCallback abstract operation
- The HostPromiseRejectionTracker abstract operation
- The HostResolveImportedModule 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 IsSharedArrayBuffer abstract operation
- The NewObjectEnvironment abstract operation
The NormalCompletion abstract operation
The OrdinaryGetPrototypeOf abstract operation
The OrdinarySetPrototypeOf abstract operation
The OrdinaryIsExtensible abstract operation
The OrdinaryPreventExtensions abstract operation
The OrdinaryDefineOwnProperty abstract operation
The OrdinaryGet abstract operation
The OrdinarySet abstract operation
The OrdinaryOwnPropertyKeys abstract operation
The ObjectCreate abstract operation
The ParseModule abstract operation
The ParseScript abstract operation
The NewPromiseReactionJob abstract operation
The NewPromiseResolveThenableJob abstract operation
The RegExpBuiltinExec abstract operation
The RegExpCreate abstract operation
The RunJobs abstract operation
The SameValue abstract operation
The ScriptEvaluation abstract operation
The SetImmutatablePrototype 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 Atomics object
The Date class
The RegExp class
The SharedArrayBuffer class
The Type class
The Type class
The eval() function
The [[IsHTMLDDA]] internal slot
import()
import.meta
The HostGetImportMetaProperties abstract operation
The typeof operator
The delete operator
The TypedArray Constructors table

Users agents that support JavaScript must also implement ECMAScript Internationalization API. [JSINTL] p1273

User agents that support JavaScript must also implement the Top-Level Await proposal. [JSTLA] p1273

WebAssembly
The following term is defined in WebAssembly JavaScript Interface: [WASMJS] p1277

- WebAssembly.Module

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] p1277

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] p1277, [UIEVENTS] p1277

In particular, the following features are defined in DOM: [DOM] p1277

- Attr interface
- Comment interface
- DOMImplementation interface
- Document interface
- DocumentOrShadowRoot interface
- DocumentFragment interface
- DocumentType interface
- ChildNode interface
- Element interface
- attachShadow() method.
- An element’s shadow root
- The retargeting algorithm
- Node interface
- NodeList interface
- ProcessingInstruction interface
- ShadowRoot interface
- Text interface
- node document concept
document type concept

host concept

The shadow root concept, and its delegates focus and available to element internals.

The shadow host concept

HTMLCollection interface, its length attribute, and its item() and namedItem() methods

The terms collection and represented by the collection

DOMTokenList interface, and its value attribute

createDocument() method

createHTMLDocument() method

createElement() method

createElementNS() method

getElementsByClassName() method

getElementsByTagName() method

appendChild() method

cloneNode() method

importNode() method

preventDefault() method

id attribute

setAttribute() method

textContent attribute

The tree, shadow tree, and node tree concepts

The tree order and shadow-including tree order concepts

The child concept

root concept

shadow-including root concept

The inclusive ancestor, shadow-including descendant, shadow-including inclusive descendant, and shadow-including inclusive ancestor concepts

The first child and next sibling concepts

The document element concept

The in a document tree, in a document (legacy), and connected concepts

The slot concept, and its name and assigned nodes

The assigned slot concept.

The find flattened slottables algorithm

The assign a slot algorithm

The pre-insert, insert, append, replace, replace all, string replace all, remove, and adopt algorithms for nodes

The insertion steps, removing steps, adopting steps, and children changed steps hooks for elements

The change, append, remove, replace, and set value algorithms for attributes

The attribute change steps hook for attributes

The attribute list concept

The data of a text node

The child text content of a node

The descendant text content of a node

Event interface

Event and derived interfaces constructor behavior

EventTarget interface

The activation behavior hook

The legacy-pre-activation behavior hook

The legacy-canceled-activation behavior hook

The create an event algorithm

The fire an event algorithm

The canceled flag

The dispatch algorithm

EventInit dictionary type

type attribute

target attribute

currentTarget attribute

bubbles attribute

cancelable attribute

composed attribute

composed flag

isTrusted attribute

initEvent() method

add an event listener

addEventListener() method

The remove an event listener and remove all event listeners algorithms

EventListener callback interface

The type of an event

An event listener and its type and callback

The encoding (herein the character encoding), mode, and content type of a Document

The distinction between XML documents and HTML documents

The 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 concept of an element's classes

The term supported token concept

The concept of a DOM range, and the terms start, end, and boundary point as applied to ranges.

The create an element algorithm

The element interface concept

The concepts of custom element state, and of defined and custom elements

An element's namespace, namespace prefix, local name, custom element definition, and is value

MutationObserver interface and mutation observers in general
The following features are defined in *UI Events*: [UIEVENTS]^{p1277}

- The `MouseEvent` interface
- The `MouseEvent` interface's `relatedTarget` attribute
- The `MouseEventInit` dictionary type
- The `FocusEvent` interface
- The `FocusEvent` interface's `relatedTarget` attribute
- The `UIEvent` interface
- The `UIEvent` interface's `view` attribute
- `auxclick` event
- `click` event
- `dblclick` event
- `mousedown` event
- `mouseenter` event
- `mouseleave` event
- `mousemove` event
- `mouseout` event
- `mouseover` event
- `mouseup` event
- `wheel` event
- `keydown` event
- `keypress` event
- `keyup` event

The following features are defined in *Touch Events*: [TOUCH]^{p1277}

- `Touch` interface
- `Touch point` concept
- `touchend` event

The following features are defined in *Pointer Events*: [POINTEREVENTS]^{p1275}

- `PointerEvent` interface
- `pointerup` event

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 *DOM Parsing and Serialization*: [DOMPARSING]^{p1272}

- `innerHTML`
- `outerHTML`

The following features are defined in *Selection API*: [SELECTION]^{p1276}

- `selection`
- `Selection`

**Note**

*User agents are encouraged to implement the features described in execCommand.* [EXECCOMMAND]^{p1272}

The following parts of *Fullscreen API* are referenced from this specification, in part to define the rendering of *dialog* elements, and also to define how the Fullscreen API interacts with HTML: [FULLSCREEN]^{p1273}

- `top layer` (an ordered set) and its `add` operation
- `requestFullscreen()`
- `run the fullscreen steps`

*High Resolution Time* provides the *current high resolution time* and the `DOMHighResTimeStamp` typedef. [HRT]^{p1273}

**File API**

This specification uses the following features defined in *File API*: [FILEAPI]^{p1272}

- The `Blob` interface and its `type` attribute
- The `File` interface and its `name` and `lastModified` attributes
- The `FileList` interface
- The concept of a `Blob's` snapshot state
- The concept of read errors
- `Blob URL Store`
- `Blob URL entry` and its `object`

**Indexed Database API**

This specification uses cleanup *Indexed Database transactions* defined by Indexed Database API. [INDEXEDDB]^{p1273}
Media Source Extensions

The following terms are defined in Media Source Extensions: [MEDIASOURCE]31274

• MediaSource interface
• detaching from a media element

Media Capture and Streams

The following terms are defined in Media Capture and Streams: [MEDIASTREAM]31274

• MediaStream interface

Reporting

The following terms are defined in Reporting: [REPORTING]31274

• Queue a report
• report type
• visible to ReportingObservers

XMLHttpRequest

The following features and terms are defined in XMLHttpRequest: [XHR]31278

• The XMLHttpRequest interface, and its responseXML attribute
• The ProgressEvent interface, and its lengthComputable, loaded, and total attributes
• The FormData interface
• The FormDataEntryValue type
• entry
• create an entry

Battery Status

The following features are defined in Battery Status API: [BATTERY]31270

• getBattery() method

Media Queries

Implementations must support Media Queries. The <media-condition> feature is defined therein. [MQ]31274

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.

When this specification requires that something be parsed according to a particular CSS grammar, the relevant algorithm in CSS Syntax must be followed, including error handling rules. [CSSSYNTAX]31272

Example

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.

To parse a CSS <color> value, given a string input with an optional element element, run these steps:

1. Let color be the result of parsing input as a CSS <color>.[CSSCOLOR]31271
2. If color is failure, then return failure.
3. If color is 'currentcolor', then:
   1. If element is not given, then set color to opaque black.
   2. Otherwise, set color to the computed value of the 'color' property of element.
4. Return color.

The following terms and features are defined in Cascading Style Sheets (CSS): [CSS]31270

• viewport
• line box
• out-of-flow
• in-flow
• collapsing margins
The basic version of the `display` property is defined in CSS, and the property is extended by other CSS modules. [CSS][p1270] [CSSRUBY][p1272] [CSSTABLE][p1272]

The following terms and features are defined in CSS Box Model: [CSSBOX][p1271]

- **content area**
- **content box**
- **border box**
- **margin box**
- **border edge**
- **margin edge**

The `margin-top`, `margin-bottom`, `margin-left`, and `margin-right` properties

The `padding-top`, `padding-bottom`, `padding-left`, and `padding-right` properties

The following features are defined in CSS Logical Properties: [CSSLOGICAL][p1271]

- The `margin-block-start`, `margin-block-end`, `margin-inline-start`, and `margin-inline-end` properties
- The `padding-block-start`, `padding-block-end`, `padding-inline-start`, and `padding-inline-end` properties
- The `border-block-start-width` property
- The `block-size` property
- The `inline-size` property
- The `inset-block-start` property
- The `inset-block-end` property

The following terms are defined in CSS Color: [CSSCOLOR][p1271]

- **named color**
- `<color>`
- The `color` property
- The `currentcolor` value
- `opaque black`
- `transparent black`

The following terms are defined in CSS Images: [CSSIMAGES][p1271]

- **default object size**
- **intrinsic dimensions**
- **intrinsic aspect ratio**
- **intrinsic height**
- **intrinsic width**
- The `image-orientation` property
- The `object-fit` property

The term **paint source** is used as defined in CSS Images Level 4 to define the interaction of certain HTML elements with the CSS `element()` function. [CSSIMAGES4][p1277]

The following features are defined in CSS Backgrounds and Borders: [CSSBG][p1271]

- The `background-color` property
- The `background-image` property
- The `border-radius` property

**CSS Backgrounds and Borders** also defines the following border properties: [CSSBG][p1271]
The following features are defined in CSS Box Alignment: [CSSALIGN]\(^{1270}\)

- The ‘align-content’ property
- The ‘align-items’ property
- The ‘align-self’ property
- The ‘justify-self’ property
- The ‘justify-content’ property
- The ‘justify-items’ property

The following terms and features are defined in CSS Display: [CSSDISPLAY]\(^{1271}\)

- outer display type
- block-level
- block container
- formatting context
- block formatting context
- inline formatting context
- replaced element
- CSS box

The following features are defined in CSS Flexible Box Layout: [CSSFLEXBOX]\(^{1271}\)

- The ‘flex-direction’ property
- The ‘flex-wrap’ property

The following terms and features are defined in CSS Fonts: [CSSFONTS]\(^{1271}\)

- first available font
- The ‘font-family’ property
- The ‘font-weight’ property
- The ‘font-size’ property
- The ‘font’ property

The following features are defined in CSS Grid Layout: [CSSGRID]\(^{1271}\)

- The ‘grid-auto-columns’ property
- The ‘grid-auto-flow’ property
- The ‘grid-auto-rows’ property
- The ‘grid-column-gap’ property
- The ‘grid-row-gap’ property
- The ‘grid-template-areas’ property
- The ‘grid-template-columns’ property
- The ‘grid-template-rows’ property

The following terms are defined in CSS Inline Layout: [CSSINLINE]\(^{1271}\)

- alphabetic baseline
- ascent metric
- descent metric
- hanging baseline
- ideographic-under baseline

The following terms and features are defined in CSS Intrinsic & Extrinsic Sizing: [CSSSIZING]\(^{1272}\)

- fit-content inline size

The ‘list-style-type’ property is defined in CSS Lists and Counters. [CSSLISTS]\(^{1271}\)

The following features are defined in CSS Overflow. [CSSOVERFLOW]\(^{1271}\)

- The ‘overflow’ property and its ‘hidden’ value
- The ‘text-overflow’ property

The following terms and features are defined in CSS Positioned Layout: [CSSPOSITION]\(^{1272}\)

- absolutely-positioned
- The ‘position’ property and its ‘static’ value

The following features are defined in CSS Multi-column Layout. [CSSMULTICOL]\(^{1271}\)
The `column-count` property
The `column-fill` property
The `column-gap` property
The `column-rule` property
The `column-width` property

The `ruby-base` value of the `display` property is defined in CSS Ruby Layout. [CSSRUBY]p1272

The following features are defined in CSS Table: [CSSTABLE]p1272
- The `border-spacing` property
- The `border-collapse` property
- The `table-cell`, `table-row`, `table-caption`, and `table` values of the `display` property

The following features are defined in CSS Text: [CSSTEXT]p1272
- The `text-transform` property
- The `white-space` property
- The `text-align` property
- The `letter-spacing` property

The following features are defined in CSS Writing Modes: [CSSWM]p1272
- The `direction` property
- The `unicode-bidi` property
- The `block flow direction`, `block size`, `inline size`, `block-start`, `block-end`, `inline-start`, `inline-end`, `line-left`, and `line-right` concepts

The following features are defined in CSS Basic User Interface: [CSSUI]p1272
- The `outline` property
- The `cursor` property
- The `appearance` property

The algorithm to update animations and send events is defined in Web Animations. [WEBANIMATIONS]p1277.

Implementations that support scripting must support the CSS Object Model. The following features and terms are defined in the CSSOM specifications: [CSSOM]p1272, [CSSOMVIEW]p1272

Screen interface
LinkStyle interface
CSSStyleDeclaration interface
style IDL attribute
cssText attribute of CSSStyleDeclaration
StyleSheet interface
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

CSS style sheet set
CSS style sheet set name
preferred CSS style sheet set name
change the preferred CSS style sheet set name
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
set up browsing context features

The following features and terms are defined in CSS Syntax: [CSSSYNTAX]p1272

conformant style sheet
parse a list of component values
parse a comma-separated list of component values
The following terms are defined in Selectors: [SELECTORS]\(^{p1276}\)

- **type selector**
- **attribute selector**
- **pseudo-class**

The following features are defined in CSS Values and Units: [CSSVALUES]\(^{p1272}\)

- `<length>`
- The ‘em’ unit
- The ‘ex’ unit
- The ‘vw’ unit
- The ‘vmin’ unit
- The ‘vmax’ unit
- The ‘attr()’ function
- The **math functions**

The term **style attribute** is defined in CSS Style Attributes. [CSSATTR]\(^{p1271}\)

The following terms are defined in the CSS Cascading and Inheritance: [CSSCASCADE]\(^{p1271}\)

- **specified value**
- **computed value**
- **used value**

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. [CSSFONTS]\(^{p1271}\) [CSSFONTLOAD]\(^{p1271}\)

The following interfaces and terms are defined in Geometry Interfaces: [GEOMETRY]\(^{p1273}\)

- **DOMMatrix** interface, and associated **m11 element**, **m12 element**, **m21 element**, **m22 element**, **m41 element**, and **m42 element**
- **DOMMatrix2DInit** and **DOMMatrixInit** dictionaries
- The **create a DOMMatrix from a dictionary** and **create a DOMMatrix from a 2D dictionary** algorithms for **DOMMatrix2DInit** or **DOMMatrixInit**

The following terms are defined in the CSS Scoping: [CSSSCOPING]\(^{p1277}\)

- **flat tree**

The following terms and features are defined in CSS Color Adjustment: [CSSCOLORADJUST]\(^{p1271}\)

- **'color-scheme'**

The following term is defined in CSS Pseudo-Elements: [CSSPSEUDO]\(^{p1272}\)

- **'::file-selector-button'**

**Intersection Observer**

The following term is defined in Intersection Observer: [INTERSECTIONOBSERVER]\(^{p1273}\)

- **run the update intersection observations steps**
- **IntersectionObserver**
- **IntersectionObserverInit**
- **observe**
- **unobserve**
- **isIntersecting**
- **target**

**WebGL**

The following interfaces are defined in the WebGL specifications: [WEBGL]\(^{p1277}\)

- **WebGLRenderingContext** interface
- **WebGL2RenderingContext** interface
- **WebGLContextAttributes** dictionary

**WebVTT**

Implementations may support WebVTT as a text track format for subtitles, captions, metadata, etc., for media resources. [WEBVTT]\(^{p1277}\)

The following terms, used in this specification, are defined in WebVTT:
The WebSocket protocol
The following terms are defined in Fetch: [FETCH]

- establish a WebSocket connection

The following terms are defined in The WebSocket protocol: [WSP]

- the WebSocket connection is established
- extensions in use
- subprotocol in use
- 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 Accessible Rich Internet Applications (ARIA), as are the following roles: [ARIA]

- button
- presentation

In addition, the following aria-* content attributes are defined in ARIA: [ARIA]

- aria-checked
- aria-describedby
- aria-disabled
- aria-label

Finally, the following terms are defined ARIA: [ARIA]

- role
- accessible name

The ARIAMixin interface, with its associated ARIAMixin getter steps and ARIAMixin setter steps hooks

Content Security Policy
The following terms are defined in Content Security Policy: [CSP]

- Content Security Policy
disposition
directive set
Content Security Policy directive
CSP list
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 Should navigation request of type be blocked by Content Security Policy? algorithm
The Should navigation response to navigation request of type in target be blocked by Content Security Policy? algorithm
The report-uri directive
The EnsureCSPDoesNotBlockStringCompilation abstract operation
The Is base allowed for Document? algorithm
The frame-ancestors directive
The sandbox directive
The contains a header-delivered Content Security Policy property.

Service Workers
The following terms are defined in Service Workers: [SW]

- active worker
- client message queue
control
Secure Contexts
The following algorithms are defined in Secure Contexts: [SECURE-CONTEXTS]¹²⁷⁶

- **Is url potentially trustworthy?**

Permissions Policy
The following terms are defined in Permissions Policy: [PERMISSIONSPOLICY]²¹²⁷

- **permissions policy**
- **policy-controlled feature**
- **container policy**
- **serialized permissions policy**
- **default allowlist**
- **The creating a permissions policy algorithm**
- **The creating a permissions policy from a response algorithm**
- **The is feature enabled by policy for origin algorithm**
- **The process permissions policy attributes algorithm**

Payment Request API
The following feature is defined in Payment Request API: [PAYMENTREQUEST]²¹²⁷

- **PaymentRequest interface**

MathML
While support for MathML as a whole is not required by this specification (though it is encouraged, at least for web browsers), certain features depend upon small parts of MathML being implemented. [MATHML]²¹²⁶

The following features are defined in Mathematical Markup Language (MathML):

- **MathML annotation-xml element**
- **MathML math element**
- **MathML merror element**
- **MathML mi element**
- **MathML mn element**
- **MathML mo element**
- **MathML mtext element**

SVG
While support for SVG as a whole is not required by this specification (though it is encouraged, at least for web browsers), certain features depend upon parts of SVG being implemented.

User agents that implement SVG must implement the SVG 2 specification, and not any earlier revisions.

The following features are defined in the SVG 2 specification: [SVG]²¹²⁶

- **SVGEElement interface**
- **SVGImageElement interface**
- **SVGScriptElement interface**
- **SVGSVGEElement interface**
- **SVG a element**
- **SVG desc element**
- **SVG foreignObject element**
- **SVG image element**
- **SVG script element**
- **SVG svg element**
- **SVG title element**
- **SVG use element**

Filter Effects
The following feature is defined in Filter Effects: [FILTERS]²¹²²

- **<filter-value-list>**

Cooperative Scheduling of Background Tasks
The following features are defined in Cooperative Scheduling of Background Tasks: [REQUESTIDLECALLBACK]²¹²⁷
Storage

The following terms are defined in Storage: [STORAGE][p1276]

- obtain a local storage bottle map
- obtain a session storage bottle map
- storage proxy map
- legacy-clone a browsing session storage shed

Web App Manifest

The following features are defined in Web App Manifest: [MANIFEST][p1274]

- application manifest
- installed web application
- process the manifest

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 HTTP State Management Mechanism (Cookies) as well. [HTTP][p1273] [COOKIES][p1270]

Note

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

2.1.10 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.

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

For example, while strongly discouraged from doing so, an implementation could add a new IDL attribute "typeTime" to a control 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 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 that 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.

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.

### 2.1.11 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. [XPATH10]

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.) [XSLT10]
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 inline, etc.).

2.2 Policy-controlled features

This document defines the following policy-controlled features:

- "autoplay", which has a default allowlist of 'self'.
- "cross-origin-isolated", which has a default allowlist of 'self'.
- "document-domain", which has a default allowlist of '*'.

2.3 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.

2.3.1 Common parser idioms

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.

2.3.2 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
Here is an example of a checkbox that is checked and disabled. The `checked` and `disabled` attributes are the boolean attributes.

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

This could be equivalently written as this:

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

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

```html
<label><input type='checkbox' checked name=cheese disabled=**) Cheese</label>
```

### 2.3.3 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, 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.*

### 2.3.4 Numbers

#### 2.3.4.1 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 ASCII whitespace** within `input` given `position`.
5. If `position` is past the end of `input`, return an error.
6. If the character indicated by \texttt{position} (the first character) is a U+002D HYPHEN-MINUS character (-):
   
   1. Let \texttt{sign} be "negative".
   2. Advance \texttt{position} to the next character.
   3. If \texttt{position} is past the end of \texttt{input}, return an error.

   Otherwise, if the character indicated by \texttt{position} (the first character) is a U+002B PLUS SIGN character (+):
   
   1. Advance \texttt{position} to the next character. (The "+" is ignored, but it is not conforming.)
   2. If \texttt{position} is past the end of \texttt{input}, return an error.

7. If the character indicated by \texttt{position} is not an ASCII digit, then return an error.

8. Collect a sequence of code points that are ASCII digits from \texttt{input} given \texttt{position}, and interpret the resulting sequence as a base-ten integer. Let \texttt{value} be that integer.

9. If \texttt{sign} is "positive", return \texttt{value}, otherwise return the result of subtracting \texttt{value} from zero.

2.3.4.2 Non-negative integers \textsuperscript{69}

A string is a valid non-negative integer if it consists of one or more ASCII digits.

A valid non-negative integer\textsuperscript{69} 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 \texttt{input} be the string being parsed.
2. Let \texttt{value} be the result of parsing \texttt{input} using the rules for parsing integers\textsuperscript{68}.
3. If \texttt{value} is an error, return an error.
4. If \texttt{value} is less than zero, return an error.
5. Return \texttt{value}.

2.3.4.3 Floating-point numbers \textsuperscript{69}

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\textsuperscript{69} 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.

\textbf{Note}

The Infinity and Not-a-Number (NaN) values are not valid floating-point numbers\textsuperscript{69}.
The best representation of the number \( n \) as a floating-point number is the string obtained from running \( \text{ToString}(n) \). The abstract operation \( \text{ToString} \) is not uniquely determined. When there are multiple possible strings that could be obtained from \( \text{ToString} \) 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 \( \text{input} \) be the string being parsed.
2. Let \( \text{position} \) be a pointer into \( \text{input} \), initially pointing at the start of the string.
3. Let \( \text{value} \) have the value 1.
4. Let \( \text{divisor} \) have the value 1.
5. Let \( \text{exponent} \) have the value 1.
6. Skip ASCII whitespace within \( \text{input} \) given \( \text{position} \).
7. If \( \text{position} \) is past the end of \( \text{input} \), return an error.
8. If the character indicated by \( \text{position} \) is a U+002D HYPHEN-MINUS character (-):
   1. Change \( \text{value} \) and \( \text{divisor} \) to -1.
   2. Advance \( \text{position} \) to the next character.
   3. If \( \text{position} \) is past the end of \( \text{input} \), return an error.
Otherwise, if the character indicated by \( \text{position} \) (the first character) is a U+002B PLUS SIGN character (+):
   1. Advance \( \text{position} \) to the next character. (The "+" is ignored, but it is not conforming.)
   2. If \( \text{position} \) is past the end of \( \text{input} \), return an error.
9. If the character indicated by \( \text{position} \) is a U+002E FULL STOP (.), and that is not the last character in \( \text{input} \), and the character after the character indicated by \( \text{position} \) is an ASCII digit, then set \( \text{value} \) to zero and jump to the step labeled fraction.
10. If the character indicated by \( \text{position} \) is not an ASCII digit, then return an error.
11. Collect a sequence of code points that are ASCII digits from \( \text{input} \) given \( \text{position} \), and interpret the resulting sequence as a base-ten integer. Multiply \( \text{value} \) by that integer.
12. If \( \text{position} \) is past the end of \( \text{input} \), jump to the step labeled conversion.
13. Fraction: If the character indicated by \( \text{position} \) is a U+002E FULL STOP (.), run these substeps:
   1. Advance \( \text{position} \) to the next character.
   2. If \( \text{position} \) is past the end of \( \text{input} \), or if the character indicated by \( \text{position} \) 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 \( \text{position} \) 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 \( \text{divisor} \) by ten.
   5. Add the value of the character indicated by \( \text{position} \), interpreted as a base-ten digit (0..9) and divided by \( \text{divisor} \), to \( \text{value} \).
   6. Advance \( \text{position} \) to the next character.
   7. If \( \text{position} \) is past the end of \( \text{input} \), 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 U+0065 (e) or a U+0045 (E), then:
   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.
   4. If the character indicated by position is not an ASCII digit, then jump to the step labeled conversion.
   5. Collect a sequence of code points that are ASCII digits from input given position, and interpret the resulting sequence as a base-ten integer. Multiply exponent by that integer.
   6. Multiply value by ten raised to the exponentth 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.

2.3.4.4 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 failure; 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 position variable for input, initially pointing at the start of input.
3. Skip ASCII whitespace within input given position.
4. If position is past the end of input or the code point at position within input is not an ASCII digit, then return failure.
5. Collect a sequence of code points that are ASCII digits from input given position, and interpret the resulting sequence as a base-ten integer. Let value be that number.
6. If position is past the end of input, then return value as a length.
7. If the code point at position within input is U+002E (.), then:
   1. Advance position by 1.
   2. If position is past the end of input or the code point at position within input is not an ASCII digit, then return the current dimension value with value, input, and position.
   3. Let divisor have the value 1.
   4. While true:
1. Multiply *divisor* by ten.

2. Add the value of the code point at *position* within *input*, interpreted as a base-ten digit (0..9) and divided by *divisor*, to *value*.

3. Advance *position* by 1.

4. If *position* is past the end of *input*, then return *value* as a length.

5. If the code point at *position* within *input* is not an ASCII digit, then break.

8. Return the current dimension value with *value*, *input*, and *position*.

The **current dimension value**, given *value*, *input*, and *position*, is determined as follows:

1. If *position* is past the end of *input*, then return *value* as a length.

2. If the code point at *position* within *input* is U+0025 (%), then return *value* as a percentage.

3. Return *value* as a length.

**2.3.4.5 Non-zero percentages and lengths**

The **rules for parsing nonzero 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.

**2.3.4.6 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 ASCII whitespace). 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 code points that are ASCII whitespace, U+002C COMMA, or U+003B SEMICOLON characters from *input* given *position*. This skips past any leading delimiters.

5. While *position* is not past the end of *input*:

   1. Collect a sequence of code points that are not ASCII whitespace, U+002C COMMA, U+003B SEMICOLON, ASCII digits, U+002E FULL STOP, or U+002D HYPHEN-MINUS characters from *input* given *position*. This skips past leading garbage.

   2. Collect a sequence of code points that are not ASCII whitespace, U+002C COMMA, or U+003B SEMICOLON characters from *input* given *position*, 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 \textit{number} is an error, set \textit{number} to zero.
5. Append \textit{number} to \textit{numbers}.
6. Collect a sequence of code points that are ASCII whitespace, U+002C COMMA, or U+003B SEMICOLON characters from \textit{input} given \textit{position}. This skips past the delimiter.
6. Return \textit{numbers}.

\textbf{2.3.4.7 Lists of dimensions} \hspace{1cm} §73

The \textbf{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 \textit{percentage}, \textit{relative}, and \textit{absolute}.

1. Let \textit{raw input} be the string being parsed.
2. If the last character in \textit{raw input} is a U+002C COMMA character (,), then remove that character from \textit{raw input}.
3. \textbf{Split the string raw input on commas}. Let \textit{raw tokens} be the resulting list of tokens.
4. Let \textit{result} be an empty list of number/unit pairs.
5. For each token in \textit{raw tokens}, run the following substeps:
   1. Let \textit{input} be the token.
   2. Let \textit{position} be a pointer into \textit{input}, initially pointing at the start of the string.
   3. Let \textit{value} be the number 0.
   4. Let \textit{unit} be \textit{absolute}.
   5. If \textit{position} is past the end of \textit{input}, set \textit{unit} to \textit{relative} and jump to the last substep.
   6. If the character at \textit{position} is an ASCII digit, collect a sequence of code points that are ASCII digits from \textit{input} given \textit{position}, interpret the resulting sequence as an integer in base ten, and increment \textit{value} by that integer.
   7. If the character at \textit{position} is U+002E (.), then:
      1. Collect a sequence of code points consisting of ASCII whitespace and ASCII digits from \textit{input} given \textit{position}. Let \textit{s} be the resulting sequence.
      2. Remove all ASCII whitespace in \textit{s}.
      3. If \textit{s} is not the empty string, then:
         1. Let \textit{length} be the number of characters in \textit{s} (after the spaces were removed).
         2. Let \textit{fraction} be the result of interpreting \textit{s} as a base-ten integer, and then dividing that number by $10^{\textit{length}}$.
         3. Increment \textit{value} by \textit{fraction}.
   8. \textbf{Skip ASCII whitespace} within \textit{input} given \textit{position}.
   9. If the character at \textit{position} is a U+0025 PERCENT SIGN character (%), then set \textit{unit} to \textit{percentage}.
      Otherwise, if the character at \textit{position} is a U+002A ASTERISK character (*), then set \textit{unit} to \textit{relative}.
   10. Add an entry to \textit{result} consisting of the number given by \textit{value} and the unit given by \textit{unit}.
6. Return the list \textit{result}.

\textbf{2.3.5 Dates and times} \hspace{1cm} §73

In the algorithms below, the \textbf{number of days in month month of year year} is: 31 if \textit{month} is 1, 3, 5, 7, 8, 10, or 12; 30 if \textit{month} is 4, 6, 9, or 11; 29 if \textit{month} is 2 and \textit{year} is a number divisible by 400, or if \textit{year} is a number divisible by 4 but not by 100; and 28
otherwise. This takes into account leap years in the Gregorian calendar. \[\text{[GREGORIAN]}^{\text{p1273}}\]

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. \[\text{[ISO8601]}^{\text{p1273}}\]

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. \[\text{[GREGORIAN]}^{\text{p1273}}\]

Note

The use of the Gregorian calendar as the wire format in this specification is an arbitrary choice resulting from the cultural biases of those involved in the decision. See also the section discussing date, time, and number formats in forms (for authors), implementation notes regarding localization of form controls, and the time element.

2.3.5.1 Months \[\text{p74}\]

A month consists of a specific proleptic-Gregorian date with no time-zone information and no date information beyond a year and a month. \[\text{[GREGORIAN]}^{\text{p1273}}\]

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

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 code points that are ASCII digits from input given position. 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 code points that are ASCII digits from input given position. 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 $1 \leq \text{month} \leq 12$, then fail.
6. Return year and month.
A date consists of a specific proleptic-Gregorian date with no time-zone information, consisting of a year, a month, and a day. A string is a valid date string representing a year, month, and 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 ≤ day ≤ maxday where maxday is the number of days in the month and 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, month, and day.
6. Return date.

A yearless date consists of a Gregorian month and a day within that month, but with no associated year. A string is a valid yearless date string representing a month and a 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, in the range 1 ≤ month ≤ 12
3. A U+002D HYPHEN-MINUS character (-)
4. Two ASCII digits, representing day, in the range 1 ≤ day ≤ maxday where maxday is the number of days in the month and any arbitrary leap year (e.g. 4 or 2000)

In other words, if the month is "02", meaning February, then the day can be 29, as if the year was a leap year.
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 code points that are U+002D HYPHEN-MINUS characters (-) from input given position. If the collected sequence is not exactly zero or two characters long, then fail.
2. Collect a sequence of code points that are ASCII digits from input given position. 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 ≤ month ≤ 12, then fail.
4. Let maxday be the number of days in 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 code points that are ASCII digits from input given position. 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 ≤ day ≤ maxday, then fail.
8. Return month and day.

2.3.5.4 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, a minute, and a second if it consists of the following components in the given order:

1. Two ASCII digits, representing hour, in the range 0 ≤ hour ≤ 23
2. A U+003A COLON character (:)
3. Two ASCII digits, representing minute, in the range 0 ≤ minute ≤ 59
4. If second is nonzero, 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 ≤ s ≤ 59
   3. If second is not an integer, or optionally if second is an integer:
      1. A U+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.

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 code points that are ASCII digits from input given position. 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 ≤ hour ≤ 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 code points that are ASCII digits from input given position. 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 ≤ minute ≤ 59, then fail.

6. Let second be 0.

7. If position is not beyond the end of input and the character at position is U+003A (:), then:
   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 code points that are either ASCII digits or U+002E FULL STOP characters from input given position. 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, interpret the resulting sequence as a base-ten number (possibly with a fractional part). Set second to that number.
   4. If second is not a number in the range 0 ≤ second < 60, then fail.

8. Return hour, minute, and second.

2.3.5.5 Local dates and times

A local 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.

A string is a valid local 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 local 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 local 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, month, and day.
8. Let time be the time with hour, minute, and second.
9. Return date and time.

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 ≤ hour ≤ 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 ≤ minute ≤ 59

Note

This format allows for time-zone offsets from -23:59 to +23:59. Right now, in practice, 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, since time zones are used as political footballs and are thus subject to very whimsical policy decisions.

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 timezonehours be 0.
   2. Let timezoneminutes 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 code points that are ASCII digits from input given position. Let s be the collected sequence.
4. If s is exactly two characters long, then:
   1. Interpret s as a base-ten integer. Let that number be the timezonehours.
   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 code points that are ASCII digits from input given position. 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 timezoneminutes.

If s is exactly four characters long, then:

1. Interpret the first two characters of s as a base-ten integer. Let that number be the timezonehours.
2. Interpret the last two characters of s as a base-ten integer. Let that number be the timezoneminutes.

Otherwise, fail.

5. If timezonehours is not a number in the range 0 ≤ timezonehours ≤ 23, then fail.
6. If sign is "negative", then negate timezonehours.
7. If timezoneminutes is not a number in the range 0 ≤ timezoneminutes ≤ 59, then fail.
8. If sign is "negative", then negate timezoneminutes.

Otherwise, fail.

2. Return timezonehours and timezoneminutes.

2.3.5.7 Global dates and times §79

A global date and time consists of a specific proleptic-Gregorian date⁹⁷⁴, 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]¹²⁷³

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 string⁹⁷⁵ representing the date
2. A U+0054 LATIN CAPITAL LETTER T character (T) or a U+0020 SPACE character
3. A valid time string⁹⁷⁶ representing the time
4. A valid time-zone offset string⁹⁷⁸ 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 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**

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 datatime 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 saving time.

---

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 *timezonehours* and *timezoneminutes*. 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* month, *day* day, *hour* hours, *minute* minute, *second* second, subtracting *timezonehours* hours and *timezoneminutes* minutes. That moment in time is a moment in the UTC time zone.
10. Let *timezone* be *timezonehours* hours and *timezoneminutes* minutes from UTC.
11. Return *time* and *timezone*.
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.

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.

---

2.3.5.8 Weeks

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 ≤ week ≤ maxweek, where maxweek is the week number of the last day of week-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 code points that are ASCII digits from input given position. 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, then fail. Otherwise, move position forwards one character.
7. Collect a sequence of code points that are ASCII digits from input given position. 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 ≤ week ≤ 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.
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 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, representing 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 ASCII whitespace.
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 ASCII whitespace.
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
     Weeks. The scale is 604800.
   - U+0044 LATIN CAPITAL LETTER D character
     Days. The scale is 86400.
   - U+0048 LATIN CAPITAL LETTER H character
     Hours. The scale is 3600.
Minutes. The scale is 60.

Seconds. The scale is 1.

6. Zero or more ASCII whitespace.

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 ASCII whitespace within input given position.

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 ASCII whitespace within input given position.

8. While true:

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.
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 ASCII whitespace within input given position, and continue.
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 code points that are ASCII digits from input given position, 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 U+002E (.), then:

   1. Collect a sequence of code points that are ASCII digits from input given position. 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 ASCII whitespace within input given position.

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:

1. If next character is ASCII whitespace, then skip ASCII whitespace within input given position, 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:

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 ASCII whitespace within input given position.
9. If component count is zero, fail.

10. If months is not zero, fail.

11. Return the duration consisting of seconds seconds.

2.3.5.10 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

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.

2.3.6 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 **ASCII lower 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 ASCII 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. [CSSCOLOR]

Note

**CSS2 System Colors** are not recognized.

6. If **input**'s code point length is four, and the first character in **input** is U+0023 (#), and the last three characters of **input** are all **ASCII hex digits**, then:

   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 **code points** greater than U+FFFF in **input** (i.e., any characters that are not in the basic multilingual plane) with
the two-character string "00".

8. If input's code point length is greater than 128, 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 code point 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 code point length, to obtain three components. Let length be the code point length that all of those components have (one third the code point 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.

### Note

The 2D graphics context has a separate color syntax that also handles opacity.

#### 2.3.7 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 ASCII whitespace, where words consist of any string of one or more characters, none of which are ASCII whitespace.

A string containing a set of space-separated tokens may have leading or trailing ASCII whitespace.

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.

#### 2.3.8 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 ASCII whitespace, nor containing any U+002C COMMA characters (,), and optionally surrounded by ASCII whitespace.
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.

### 2.3.9 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 same tree.

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.
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 `scope`'s tree, in tree order, that has an `id` or `name` attribute whose value is `s`, or null if there is no such element.

**Note**

Although `id` attributes are accounted for when parsing, they are not used in determining whether a value is a **valid hash-name reference**. That is, a hash-name reference that refers to an element based on `id` is a conformance error (unless that element also has a `name` attribute with the same value).

### 2.3.10 Media queries

A string is a **valid media query list** if it matches the `<media-query-list>` production of Media Queries. [MQ]

A string **matches the environment** of the user if it is the empty string, a string consisting of only ASCII whitespace, or is a media query list that matches the user's environment according to the definitions given in Media Queries. [MQ]

### 2.4 URLs

#### 2.4.1 Terminology

A string is a **valid non-empty URL** if it is a **valid URL string** but it is not the empty string.

A string is a **valid URL potentially surrounded by spaces** if, after stripping leading and trailing ASCII whitespace from it, it is a valid URL string.

A string is a **valid non-empty URL potentially surrounded by spaces** if, after stripping leading and trailing ASCII 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. [ABOUT]

This specification defines the URL **about:html-kind** as a reserved, though unresolvable, **about**: URL, that is used as an identifier for kinds of media tracks. [ABOUT]

This specification defines the URL **about:srcdoc** as a reserved, though unresolvable, **about**: URL, that is used as the URL of iframe `srcdoc` documents. [ABOUT]
The fallback base URL of a Document object is the URL record obtained by running these steps:

1. If document is an iframe srcdoc document, then return the document base URL of document’s browsing context’s container document.
2. If document’s URL is about:blank, and document’s browsing context’s creator base URL is non-null, then return that creator base URL.
3. Return document’s URL.

The document base URL of a Document object is the absolute URL obtained by running these steps:

1. If there is no base element that has an href attribute in the Document, then return the Document’s fallback base URL.
2. Otherwise, return the frozen base URL of the first base element in the Document that has an href attribute, in tree order.

2.4.2 Parsing URLs

Parsing a URL is the process of taking a string and obtaining the URL record that it represents. While this process is defined in URL, the HTML standard defines a wrapper for convenience. (URL)

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 return failure.
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.

2.4.3 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 DOM):

- 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.

  For example, the CSS :link/visited pseudo-classes might have been affected.

  If the hyperlink has a ping attribute and its URL(s) are being shown to the user, then the ping attribute’s tokens should be reparsed relative to the element’s node document and the UI updated appropriately.
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

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.

2.5 Fetching resources

2.5.1 Terminology

A response whose type is "basic", "cors", or "default" is CORS-same-origin. [FETCH]

A response whose type is "opaque" or "opaqueredirect" is CORS-cross-origin.

A response's unsafe response is its internal response if it has one, and the response itself otherwise.

To create a potential-CORS request, given a `url`, `destination`, `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 `destination`, `mode` is `mode`, `credentials mode` is `credentialsMode`, and whose `use-URL-credentials flag` is set.

2.5.2 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 MIME Sniffing. [MIMESNIFF]

The computed MIME type of a resource must be found in a manner consistent with the requirements given in MIME Sniffing. [MIMESNIFF]

The rules for sniffing images specifically, the rules for distinguishing if a resource is text or binary, and the rules for sniffing audio and video specifically are also defined in MIME Sniffing. These rules return a MIME type as their result. [MIMESNIFF]

⚠ Warning!

It is imperative that the rules in MIME Sniffing 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 MIME Sniffing. [MIMESNIFF]

2.5.3 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-insensitive match for the word "charset". If no such match is found, return nothing.
3. Skip any ASCII whitespace 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 ASCII whitespace 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 ASCII whitespace or U+003B SEMICOLON character (;), or the end of s, whichever comes first.

\[\text{Note} \]
This algorithm is distinct from those in the HTTP specifications (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]

2.5.4 CORS settings attributes

A CORS settings attribute is an enumerated attribute. The following table lists the keywords and states for the attribute — the states given in the first cell of the rows with keywords give the states to which those keywords map.

<table>
<thead>
<tr>
<th>State</th>
<th>Keywords</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous</td>
<td>anonymous</td>
<td>Requests for the element will have their mode set to &quot;cors&quot; and their credentials mode set to &quot;same-origin&quot;.</td>
</tr>
<tr>
<td>Use Credentials</td>
<td>use-credentials</td>
<td>Requests for the element will have their mode set to &quot;cors&quot; and their credentials mode set to &quot;include&quot;.</td>
</tr>
</tbody>
</table>

The attribute's invalid value default is the Anonymous state, and its missing value default is the No CORS state. For the purposes of reflection, the canonical keyword for the Anonymous state is the anonymous keyword.

The majority of fetches governed by CORS settings attributes, will be done via the create a potential-CORS request algorithm.

For more modern features, where the request's mode is always "cors", certain CORS settings attributes have been repurposed to have a slightly different meaning, wherein they only impact the request's credentials mode. To perform this translation, we define the CORS settings attribute credentials mode for a given CORS settings attribute to be determined by switching on the attribute's state:

≡ No CORS
≡ Anonymous
   "same-origin"
≡ Use Credentials
   "include"
2.5.5 Referrer policy attributes

A referrer policy attribute is an enumerated attribute. Each referrer policy, including the empty string, is a keyword for this attribute, mapping to a state of the same name.

The attribute’s invalid value default and missing value default are both the empty string state.

The impact of these states on the processing model of various fetches is defined in more detail throughout this specification, in Fetch, and in Referrer Policy. [FETCH] [REFERRERPOLICY]

Note

Several signals can contribute to which processing model is used for a given fetch: a referrer policy attribute is only one of them. In general, the order in which these signals are processed are:

1. First, the presence of a noreferrer link type;
2. Then, the value of a referrer policy attribute;
3. Then, the presence of any meta element with name attribute set to referrer.
4. Finally, the ‘Referrer-Policy’ HTTP header.

2.5.6 Nonce attributes

A nonce content attribute represents a cryptographic nonce (“number used once”) which can be used by Content Security Policy to determine whether or not a given fetch will be allowed to proceed. The value is text. [CSP]

Elements that have a nonce content attribute ensure that the cryptographic nonce is only exposed to script (and not to side-channels like CSS attribute selectors) by taking the value from the content attribute, moving it into an internal slot named [[CryptographicNonce]], exposing it to script via the HTMLOrSVGElement interface mixin, and setting the content attribute to the empty string. Unless otherwise specified, the slot’s value is the empty string.

For web developers (non-normative)

```javascript
element . nonce
```

Returns the value of the element's [[CryptographicNonce]] internal slot.

Can be set, to update that slot's value.

The nonce IDL attribute must, on getting, return the value of this element's [[CryptographicNonce]]; and on setting, set this element's [[CryptographicNonce]] to the given value.

Note

Note how the setter for the nonce IDL attribute does not update the corresponding content attribute. This, as well as the below setting of the nonce content attribute to the empty string when an element becomes browsing-context connected, is meant to prevent exfiltration of the nonce value through mechanisms that can easily read content attributes, such as selectors. Learn more in issue #2369, where this behavior was introduced.

The following attribute change steps are used for the nonce content attribute:

1. If element does not include HTMLOrSVGElement, then return.
2. If localName is not nonce or namespace is not null, then return.
3. If value is null, then set element's [[CryptographicNonce]] to the empty string.
4. Otherwise, set element's [[CryptographicNonce]] to value.

Whenever an element including HTMLOrSVGElement becomes browsing-context connected, the user agent must execute the following steps on the element:

1. Let CSP list be element's shadow-including root's CSP list.
2. If CSP list contains a header-delivered Content Security Policy, and element has a nonce content attribute attr whose value is not the empty string, then:
   1. Let nonce be element's [[CryptographicNonce]]
   2. Set an attribute value for element using “nonce” and the empty string.
   3. Set element's [[CryptographicNonce]] to nonce.

Note
If element's [[CryptographicNonce]] were not restored it would be the empty string at this point.

Note
As each Document's CSP list is append-only, user agents can optimize away the contains a header-delivered Content Security Policy check by, for example, holding a flag on the Document, set during Document creation and initialization.

The cloning steps for elements that include HTML or SVGElement must set the [[CryptographicNonce]] slot on the copy to the value of the slot on the element being cloned.

2.5.7 Lazy loading attributes

A lazy loading 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.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lazy</td>
<td>Lazy</td>
<td>Used to defer fetching a resource until some conditions are met.</td>
</tr>
<tr>
<td>eager</td>
<td>Eager</td>
<td>Used to fetch a resource immediately; the default state.</td>
</tr>
</tbody>
</table>

The attribute's missing value default and invalid value default are both the Eager state.

The will lazy load element steps, given an element element, are as follows:

1. If scripting is disabled for element, then return false.

   Note
   This is an anti-tracking measure, because if a user agent supported lazy loading when scripting is disabled, it would still be possible for a site to track a user's approximate scroll position throughout a session, by strategically placing images in a page's markup such that a server can track how many images are requested and when.

2. If element's lazy loading attribute is in the Lazy state, then return true.

3. Return false.

Each img and iframe element has associated lazy load resumption steps, initially null.

Note
For img and iframe elements that will lazy load, these steps are run from the lazy load intersection observer's callback or when their lazy loading attribute is set to the Eager state. This causes the element to continue loading.

Each Document has a lazy load intersection observer, initially set to null but can be set to an IntersectionObserver instance.

To start intersection-observing a lazy loading element element, run these steps:

1. Let doc be element's node document.

2. If doc's lazy load intersection observer is null, set it to a new IntersectionObserver instance, initialized as follows:
The intention is to use the original value of the `IntersectionObserver` constructor. However, we're forced to use the JavaScript-exposed constructor in this specification, until `Intersection Observer` exposes low-level hooks for use in specifications. See bug `w3c/IntersectionObserver#427` which tracks this. [INTERSECTIONOBSERVER] 

The callback is these steps, with arguments `entries` and `observer`:

1. For each entry in `entries` using a method of iteration which does not trigger developer-modifiable array accessors or iteration hooks:
   1. Let `resumptionSteps` be null.
   2. If `entry.isIntersecting` is true, then set `resumptionSteps` to `entry.target's lazy load resumption steps`.
   3. If `resumptionSteps` is null, then return.
   4. Stop intersection-observing a lazy loading element for `entry.target`.
   5. Set `entry.target's lazy load resumption steps` to null.
   6. Invoke `resumptionSteps`.

The intention is to use the original value of the `isIntersecting` and `target` getters. See `w3c/IntersectionObserver#427`. [INTERSECTIONOBSERVER] 

The options is an `IntersectionObserverInit` dictionary with the following dictionary members: «[ "rootMargin" → lazy load root margin ]»

Note

This allows for fetching the image during scrolling, when it does not yet — but is about to — intersect the viewport.

The lazy load root margin suggestions imply dynamic changes to the value, but the `IntersectionObserver` API does not support changing the root margin. See issue `w3c/IntersectionObserver#428`.

3. Call `doc's lazy load intersection observer's observe` method with `element` as the argument.

The intention is to use the original value of the `observe` method. See `w3c/IntersectionObserver#427`. [INTERSECTIONOBSERVER] 

To stop intersection-observing a lazy loading element `element`, run these steps:

1. Let `doc` be `element's node document`.
2. Assert: `doc's lazy load intersection observer` is not null.
3. Call `doc's lazy load intersection observer unobserve` method with `element` as the argument.

The intention is to use the original value of the `unobserve` method. See `w3c/IntersectionObserver#427`. [INTERSECTIONOBSERVER] 

The lazy load root margin is an implementation-defined value, but with the following suggestions to consider:

- Set a minimum value that most often results in the resources being loaded before they intersect the viewport under normal usage patterns for the given device.
- The typical scrolling speed: increase the value for devices with faster typical scrolling speeds.
- The current scrolling speed or momentum: the UA can attempt to predict where the scrolling will likely stop, and adjust the value accordingly.
- The network quality: increase the value for slow or high-latency connections.
- User preferences can influence the value.
2.6 Common DOM interfaces

2.6.1 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 USVString attribute whose content attribute is defined to contain a URL, then on getting, if the content attribute is absent, the IDL attribute must return the empty string. Otherwise, the IDL attribute must parse the value of the content attribute relative to the element's node document and if that is successful, return the resulting URL string. If parsing fails, then the value of the content attribute must be returned instead, converted to a USVString. On setting, the content attribute must be set to the specified new 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 keyword value associated with the state the attribute is in, 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). If there are multiple keyword values for the state, then return the conforming one. If there are multiple conforming keyword values, then one will be designated the canonical keyword; choose that one. 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 state then the IDL attribute must return null, otherwise, the IDL attribute must return the keyword value associated with the state the attribute is in. If there are multiple keyword values for the state, then return the conforming one. If there are multiple conforming keyword values, then one will be designated the canonical keyword; choose that one. 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 or USVString 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" DOMException. 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

---

Note: It is important for privacy that the lazy load root margin not leak additional information. For example, the typical scrolling speed on the current device could be imprecise so as to not introduce a new fingerprinting vector.
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\textsuperscript{969}, 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" \text{DOMException}. 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\textsuperscript{969} 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 with fallback, then the behavior is similar to the previous case, but disallowed values are converted to the default value. On getting, the content attribute must first be parsed according to the rules for parsing non-negative integers\textsuperscript{969}, 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. On setting, 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; then, \( n \) must be converted to the shortest possible string representing the number as a valid non-negative integer\textsuperscript{969} 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 clamped to the range \([min, max]\), then on getting, the content attribute must first be parsed according to the rules for parsing non-negative integers\textsuperscript{969}, and if that is successful, and the value is between \( min \) and \( max \) inclusive, the resulting value must be returned. If it fails, the default value must be returned. If it succeeds but the value is less than \( min \), \( min \) must be returned. If it succeeds but the value is greater than \( max \), \( max \) must be returned. On setting, it behaves the same as setting a regular reflected unsigned integer.

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\textsuperscript{970}, 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\textsuperscript{970} 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\textsuperscript{970}, 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\textsuperscript{970} and then that string must be used as the new content attribute value.

The values Infinity and Not-a-Number (NaN) values throw an exception on setting, as defined in Web IDL. [WEBIDL]\textsuperscript{1277}

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.

2.6.2 Collections\textsuperscript{998}

The HTMLFormControlsCollection\textsuperscript{999} and HTMLOptionsCollection\textsuperscript{1000} interfaces are collections derived from the HTMLCollection interface. The HTMLAllCollection\textsuperscript{997} interface is a collection, but is not so derived.

2.6.2.1 The HTMLAllCollection\textsuperscript{997} interface\textsuperscript{998}

The HTMLAllCollection\textsuperscript{997} interface is used for the legacy document.all\textsuperscript{1234} attribute. It operates similarly to HTMLCollection; the main differences are that it allows a staggering variety of different (ab)uses of its methods to all end up returning something, and that it can be called as a function as an alternative to property access.
Objects that implement the **HTMLAllCollection** interface are legacy platform objects with an additional [[Call]] internal method described in the section below. They also have an [[IsHTMLDDA]] internal slot.

**Note**

Objects that implement the **HTMLAllCollection** interface have several unusual behaviors, due to the fact that they have an [[IsHTMLDDA]] internal slot:

- The **ToBoolean** abstract operation in JavaScript returns false when given objects implementing the **HTMLAllCollection** interface.
- The **Abstract Equality Comparison** algorithm, when given objects implementing the **HTMLAllCollection** interface, returns true when compared to the undefined and null values. (Comparisons using the **Strict Equality Comparison** algorithm, and Abstract Equality comparisons to other values such as strings or objects, are unaffected.)
- The **typeof** operator in JavaScript returns the string "undefined" when applied to objects implementing the **HTMLAllCollection** interface.

These special behaviors are 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. [JAVASCRIPT]

---

**IDL**

```idl
[Exposed=Window,
 LegacyUnenumerableNamedProperties]
interface HTMLAllCollection {
    readonly attribute unsigned long length;
    getter Element (unsigned long index);
    getter (HTMLCollection or Element)? namedItem(DOMString name);
    (HTMLCollection or Element)? item(optional DOMString nameOrIndex);

    // Note: HTMLAllCollection objects have a custom [[Call]] internal method and an [[IsHTMLDDA]]
    // internal slot.
};
```

**For web developers (non-normative)**

- `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.

  Only **button**, **form**, **iframe**, **input**, **map**, **meta**, **object**, **select**, and **textarea** elements can have a name for the purpose of this method; their name is given by the value of their `name` attribute.
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 attributes of all the elements represented by the collection, and the non-empty values of all the name attributes of all the "all"-named 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(s) from this HTMLAllCollection given name.

The item(nameOrIndex) method must perform the following steps:

1. If nameOrIndex was not provided, return null.
2. Return the result of getting the "all"-indexed or named element(s) from this HTMLAllCollection, given nameOrIndex.

The following elements are "all"-named elements: a, 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 indexth element in collection, or null if there is no such indexth element.

To get the "all"-named element(s) from an HTMLAllCollection collection given a name name, perform the following steps:

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.

To get the "all"-indexed or named element(s) from an HTMLAllCollection collection given nameOrIndex:

1. 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.
2. Return the result of getting the "all"-named element(s) from this HTMLAllCollection given nameOrIndex.

2.6.2.1.1 [[Call]] (thisArgument, argumentsList)

1. If argumentsList's size is zero, or if argumentsList[0] is undefined, return null.
2. Let nameOrIndex be the result of converting argumentsList[0] to a DOMString.
3. Let result be the result of getting the "all"-indexed or named element(s) from this HTMLAllCollection given nameOrIndex.
4. Return the result of converting result to an ECMAScript value.
2.6.2.2 The **HTMLFormControlsCollection** interface

The **HTMLFormControlsCollection** interface is used for collections of listed elements in form elements.

```idl
[Exposed=Window]
interface HTMLFormControlsCollection : HTMLCollection {
  // inherits length and item()
  getter (RadioNodeList or Element)? namedItem(DOMString name); // shadows inherited namedItem()
}
```

### For web developers (non-normative)

- **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]**

  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 `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.

2.6.2.3 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.

```idl
[Exposed=Window]
interface HTMLOptionsCollection : HTMLCollection {
  // inherits item(), namedItem()
  // inherits length, item(), namedItem()
  [CEReactions] attribute unsigned long length; // shadows inherited length
  [CEReactions] setter undefined (unsigned long index, HTMLOptionElement? option);
  [CEReactions] undefined add((HTMLOptionElement or HTMLOptGroupElement) element, optional (HTMLElement or long)? before = null);
  [CEReactions] undefined remove(long index);
  attribute long selectedIndex;
};
```

For web developers (non-normative)

`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.
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).

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.

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 to a new 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 return.
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 indexth 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" DOMException.
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" DOMException.
3. If element and before are the same element, then return.

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).

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.

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 to a new 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 return.
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 indexth 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" DOMException.
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" DOMException.
3. If element and before are the same element, then return.
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 null, let `parent` be the parent node of `reference`. Otherwise, let `parent` be the `select` element on which the `HTMLOptionsCollection` is rooted.

6. Pre-insert element into parent node before `reference`.

The `remove(index)` method must act according to the following algorithm:

1. If the number of nodes `represented by the collection` is zero, return.
2. If `index` is not a number greater than or equal to 0 and less than the number of nodes `represented by the collection`, return.
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.

2.6.3 The `DOMStringList` interface

The `DOMStringList` interface is a non-fashionable retro way of representing a list of strings.

```idl
[Exposed=(Window, Worker)]
interface DOMStringList {
    readonly attribute unsigned long length;
    getter DOMString? item(unsigned long index);
    boolean contains(DOMString string);
};
```

⚠️ Warning!

New APIs must use `sequence<DOMString>` or equivalent rather than `DOMStringList`.

For web developers (non-normative)

- `strings.length` Returns the number of strings in `strings`.
- `strings[index]` Returns the string with index `index` from `strings`.
- `strings.item(index)` Returns true if `strings` contains `string`, and false otherwise.

Each `DOMStringList` object has an associated `list`.

The supported property indices for a `DOMStringList` object are the numbers zero to the associated list's `size` minus one. If its associated list is empty, it has no supported property indices.

The `length` attribute's getter must return this `DOMStringList` object's associated list's `size`.

The `item(index)` method, when invoked, must return the indexth item in this `DOMStringList` object's associated list, or null if `index` plus one is greater than this `DOMStringList` object's associated list's `size`.

The `contains(string)` method, when invoked, must return true if this `DOMStringList` object's associated list contains `string`, and false otherwise.
2.7 Safe passing of structured data

This section uses the terminology and typographic conventions from the JavaScript specification. [JAVASCRIPT][1273]

2.7.1 Serializable objects

Serializable objects support being serialized, and later deserialized, in a way that is independent of any given JavaScript Realm. This allows them to be stored on disk and later restored, or cloned across agent and even agent cluster boundaries.

Not all objects are Serializable objects, and not all aspects of objects that are Serializable objects are necessarily preserved when they are serialized.

Platform objects can be Serializable objects if their primary interface is decorated with the [Serializable] IDL extended attribute. Such interfaces must also define the following algorithms:

### Serialization steps, taking a platform object value, a Record serialized, and a boolean forStorage

A set of steps that serializes the data in value into fields of serialized. The resulting data serialized into serialized must be independent of any JavaScript Realm.

These steps may throw an exception if serialization is not possible.

These steps may perform a sub-serialization to serialize nested data structures. They should not call StructuredSerialize directly, as doing so will omit the important memory argument.

The introduction of these steps should omit mention of the forStorage argument if it is not relevant to the algorithm.

### Deserialization steps, taking a Record serialized and a platform object value

A set of steps that deserializes the data in serialized, using it to set up value as appropriate. value will be a newly-created instance of the platform object type in question, with none of its internal data set up; setting that up is the job of these steps.

These steps may throw an exception if deserialization is not possible.

These steps may perform a sub-deserialization to deserialize nested data structures. They should not call StructuredDeserialize directly, as doing so will omit the important targetRealm and memory arguments.

It is up to the definition of individual platform objects to determine what data is serialized and deserialized by these steps. Typically the steps are very symmetric.

The [Serializable] extended attribute must take no arguments, and must only appear on an interface. It must not appear more than once on an interface.

For a given platform object, only the object's primary interface is considered during the (de)serialization process. Thus, if inheritance is involved in defining the interface, each [Serializable]-annotated interface in the inheritance chain needs to define standalone serialization steps and deserialization steps, including taking into account any important data that might come from inherited interfaces.

#### Example

Let's say we were defining a platform object Person, which had associated with it two pieces of associated data:

- a name value, which is a string;
- and a best friend value, which is either another Person instance or null

We could then define Person instances to beSerializable objects by annotating the Person interface with the [Serializable] extended attribute, and defining the following accompanying algorithms:

#### Serialization steps

1. Set serialized.[[Name]] to value's associated name value.
2. Let serializedBestFriend be the sub-serialization of value's associated best friend value.
Objects defined in the JavaScript specification are handled by the `StructuredSerializeWithTransfer` abstract operation directly.

Note

Originally, this specification defined the concept of "cloneable objects", which could be cloned from one JavaScript Realm to another. However, to better specify the behavior of certain more complex situations, the model was updated to make the serialization and deserialization explicit.

2.7.2 Transferable objects

Transferable objects support being transferred across agents. 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 can be transferable objects if their primary interface is decorated with the `[Transferable]` IDL extended attribute. Such interfaces must also define the following algorithms:

transfer steps, taking a platform object `value` and a Record `dataHolder`

A set of steps that transfers the data in `value` into fields of `dataHolder`. The resulting data held in `dataHolder` must be independent of any JavaScript Realm.

These steps may throw an exception if transferral is not possible.

transfer-receiving steps, taking a Record `dataHolder` and a platform object `value`

A set of steps that receives the data in `dataHolder`, using it to set up `value` as appropriate. `value` will be a newly-created instance of the platform object type in question, with none of its internal data set up; setting that up is the job of these steps.

These steps may throw an exception if it is not possible to receive the transfer.

It is up to the definition of individual platform objects to determine what data is transferred by these steps. Typically the steps are very symmetric.

The `[Transferable]` extended attribute must take no arguments, and must only appear on an interface. It must not appear more than once on an interface.

For a given platform object, only the object's primary interface is considered during the transferring process. Thus, if inheritance is involved in defining the interface, each `[Transferable]`-annotated interface in the inheritance chain needs to define standalone transfer steps and transfer-receiving steps, including taking into account any important data that might come from inherited interfaces.

Platform objects that are transferable objects have a `[[Detached]]` internal slot. This is used to ensure that once a platform object has been transferred, it cannot be transferred again.

Objects defined in the JavaScript specification are handled by the `StructuredSerializeWithTransfer` abstract operation directly.
The `StructuredSerializeInternal` abstract operation takes as input a JavaScript value `value` and serializes it to a Realm-independent form, represented here as a `Record`. This serialized form has all the information necessary to later deserialize into a new JavaScript value in a different Realm.

This process can throw an exception, for example when trying to serialize un-serializable objects.

1. If `memory` was not supplied, let `memory` be an empty `map`.

   **Note**
   
   The purpose of the memory map is to avoid serializing objects twice. This ends up preserving cycles and the identity of duplicate objects in graphs.

2. If `memory[value]` exists, then return `memory[value]`.

3. Let `deep` be false.

4. If `Type(value)` is Undefined, Null, Boolean, Number, BigInt, or String, then return `{ [[Type]]: "primitive", [[Value]]: value }`.

5. If `Type(value)` is Symbol, then throw a "DataCloneError" `DOMException`.

6. Let `serialized` be an uninitialized value.

7. If `value` has a `[[BooleanData]]` internal slot, then set `serialized` to `{ [[Type]]: "Boolean", [[BooleanData]]: value.[[BooleanData]] }`.

8. Otherwise, if `value` has a `[[NumberData]]` internal slot, then set `serialized` to `{ [[Type]]: "Number", [[NumberData]]: value.[[NumberData]] }`.

9. Otherwise, if `value` has a `[[BigIntData]]` internal slot, then set `serialized` to `{ [[Type]]: "BigInt", [[BigIntData]]: value.[[BigIntData]] }`.

10. Otherwise, if `value` has a `[[StringData]]` internal slot, then set `serialized` to `{ [[Type]]: "String", [[StringData]]: value.[[StringData]] }`.

11. Otherwise, if `value` has a `[[DateValue]]` internal slot, then set `serialized` to `{ [[Type]]: "Date", [[DateValue]]: value.[[DateValue]] }`.

12. Otherwise, if `value` has a `[[RegExpMatcher]]` internal slot, then set `serialized` to `{ [[Type]]: "RegExp", [[RegExpMatcher]]: value.[[RegExpMatcher]], [[OriginalSource]]: value.[[OriginalSource]], [[OriginalFlags]]: value.[[OriginalFlags]] }`.

13. Otherwise, if `value` has an `[[ArrayBufferData]]` internal slot, then:
    1. Let `size` be `value.[[ArrayBufferByteLength]]`.
    2. If `IsSharedArrayBuffer(value)` is true, then:
        1. If the `current settings object`'s cross-origin isolated capability is false, then throw a "DataCloneError" `DOMException`.

           **Note**
           
           This check is only needed when serializing (and not when deserializing) as the cross-origin isolated capability cannot change over time and a SharedArrayBuffer cannot leave an agent cluster.

        2. If `forStorage` is true, then throw a "DataCloneError" `DOMException`.
        3. Set `serialized` to `{ [[Type]]: "SharedArrayBuffer", [[ArrayBufferData]]: value.[[ArrayBufferData]], [[ArrayBufferByteLength]]: size, [[AgentCluster]]: agentCluster }`.
    3. Otherwise:
        1. If `IsDetachedBuffer(value)` is true, then throw a "DataCloneError" `DOMException`.
        2. Let `dataCopy` be `CreateByteDataBlock(size)`.

           **Note**
           
           This can throw a `RangeError` exception upon allocation failure.
3. Perform `CopyDataBlockBytes(dataCopy, 0, value.[[ArrayBufferData]], 0, size).

4. Set `serialized` to `{ [[Type]]: "ArrayBuffer", [[ArrayBufferData]]: dataCopy, [[ArrayBufferByteLength]]: size }.

14. Otherwise, if `value` has a `[[ViewedArrayBuffer]]` internal slot, then:
   1. Let `buffer` be the value of `value`'s `[[ViewedArrayBuffer]]` internal slot.
   2. Let `bufferSerialized` be `StructuredSerializeInternal(buffer, forStorage, memory)`.
   3. Assert: `bufferSerialized.[[Type]]` is "ArrayBuffer".
   4. If `value` has a `[[DataView]]` internal slot, then set `serialized` to `{ [[Type]]: "ArrayBufferView", [[Constructor]]: "DataView", [[ArrayBufferSerialized]]: bufferSerialized, [[ByteLength]]: value.[[ByteLength]], [[ByteOffset]]: value.[[ByteOffset]] }.

5. Otherwise:
   1. Assert: `value` has a `[[TypedArrayName]]` internal slot.
   2. Set `serialized` to `{ [[Type]]: "ArrayBufferView", [[Constructor]]: value.[[TypedArrayName]], [[ArrayBufferSerialized]]: bufferSerialized, [[ByteLength]]: value.[[ByteLength]], [[ByteOffset]]: value.[[ByteOffset]], [[ArrayLength]]: value.[[ArrayLength]] }.

15. Otherwise, if `value` has a `[[MapData]]` internal slot, then:
   1. Set `serialized` to `{ [[Type]]: "Map", [[MapData]]: a new empty List }.
   2. Set `deep` to true.

16. Otherwise, if `value` has a `[[SetData]]` internal slot, then:
   1. Set `serialized` to `{ [[Type]]: "Set", [[SetData]]: a new empty List }.
   2. Set `deep` to true.

17. Otherwise, if `value` has an `[[ErrorData]]` internal slot and `value` is not a platform object, then:
   1. Let `name` be `Get(value, "name")`.
   2. If `name` is not one of "Error", "EvalError", "RangeError", "ReferenceError", "SyntaxError", "TypeError", or "URIError", then set `name` to "Error".
   3. Let `valueMessageDesc` be `value.[[GetOwnProperty]]("message")`.
   4. Let `message` be undefined if `IsDataDescriptor(valueMessageDesc)` is false, and `ToString(valueMessageDesc.[[Value]])` otherwise.
   5. Set `serialized` to `{ [[Type]]: "Error", [[Name]]: name, [[Message]]: message }.
   6. User agents should attach a serialized representation of any interesting accompanying data which are not yet specified, notably the `stack` property, to `serialized`.

   **Note:** See the Error Stacks proposal for in-progress work on specifying this data. [JSERRORSTACKS]

18. Otherwise, if `value` is an Array exotic object, then:
   1. Let `valueLenDescriptor` be `OrdinaryGetOwnProperty(value, "length")`.
   2. Let `valueLen` be `valueLenDescriptor.[[Value]]`.
   3. Set `serialized` to `{ [[Type]]: "Array", [[Length]]: valueLen, [[Properties]]: a new empty List }.
   4. Set `deep` to true.

19. Otherwise, if `value` is a platform object that is a serializable object:
   1. If `value` has a `[[Detached]]` internal slot whose value is true, then throw a "DataCloneError" DOMException.
   2. Let `typeName` be the identifier of the primary interface of `value`.
3. Set `serialized` to `{ [[Type]]: typeString }.

4. Set `deep` to true.

20. Otherwise, if `value` is a platform object, then throw a "DataCloneError" DOMException.

21. Otherwise, if `isCallable(value)` is true, then throw a "DataCloneError" DOMException.

22. Otherwise, if `value` has any internal slot other than `[[Prototype]]` or `[[Extensible]]`, then throw a "DataCloneError" DOMException.

23. Otherwise, if `value` is an exotic object and `value` is not the `%Object.prototype%` intrinsic object associated with any JavaScript realm, then throw a "DataCloneError" DOMException.

Example
For instance, a [[PromiseState]] or [[WeakMapData]] internal slot.

24. Otherwise:

1. Set `serialized` to `{ [[Type]]: "Object", [[Properties]]: a new empty List }.

2. Set `deep` to true.

Note
%Object.prototype% will end up being handled via this step and subsequent steps. The end result is that its exoticness is ignored, and after deserialization the result will be an empty object (not an immutable prototype exotic object).

25. Set `memory[value]` to `serialized`.

26. If `deep` is true, then:

1. If `value` has a `[[MapData]]` internal slot, then:

   1. Let `copiedList` be a new empty List.

   2. For each `Record` { [[Key]], [[Value]] } entry of `value.[[MapData]]`:

      1. Let `copiedEntry` be a new `Record` { [[Key]], [[Value]]: entry.[[Value]] }.

      2. If `copiedEntry.[[Key]]` is not the special value empty, append `copiedEntry` to `copiedList`.

   3. For each `Record` { [[Key]], [[Value]] } entry of `copiedList`:

      1. Let `serializedKey` be ? StructuredSerializeInternal(entry.[[Key]], forStorage, memory).

      2. Let `serializedValue` be ? StructuredSerializeInternal(entry.[[Value]], forStorage, memory).

      3. Append { [[Key]]: serializedKey, [[Value]]: serializedValue } to `serialized.[[MapData]]`.

2. Otherwise, if `value` has a `[[SetData]]` internal slot, then:

   1. Let `copiedList` be a new empty List.

   2. For each `entry` of `value.[[SetData]]`:

      1. If `entry` is not the special value empty, append `entry` to `copiedList`.

   3. For each `entry` of `copiedList`:

      1. Let `serializedEntry` be ? StructuredSerializeInternal(entry, forStorage, memory).

      2. Append `serializedEntry` to `serialized.[[SetData]]`.

3. Otherwise, if `value` is a platform object that is a serializable object, then perform the serialization steps for `value`'s primary interface, given `value`, `serialized`, and `forStorage`.

   The serialization steps may need to perform a sub-serialization. This is an operation which takes as input a
value subValue, and returns $\text{StructuredSerializeInternal}^{p105}(\text{subValue, forStorage, memory})$. (In other words, a sub-serialization$^{p107}$ is a specialization of $\text{StructuredSerializeInternal}^{p105}$ to be consistent within this invocation.)

4. Otherwise, for each key in $\text{EnumerableOwnPropertyNames}(\text{value, key})$:

   1. If $\text{HasOwnProperty}(\text{value, key})$ is true, then:
      
         1. Let $\text{inputValue}$ be $\text{value}.\sget{key, value}$.
         2. Let $\text{outputValue}$ be $\text{StructuredSerializeInternal}^{p105}(\text{inputValue, forStorage, memory})$.
         3. Append $\{[[\text{Key}]]: \text{key}, [[\text{Value}]]: \text{outputValue}\}$ to $\text{serialized}.[[\text{Properties}]]$.

27. Return $\text{serialized}$.

Example

It's important to realize that the Records produced by $\text{StructuredSerializeInternal}^{p105}$ might contain "pointers" to other records that create circular references. For example, when we pass the following JavaScript object into $\text{StructuredSerializeInternal}^{p105}$:

```javascript
const o = {};
o.myself = o;
```

it produces the following result:

```javascript
{
[[Type]]: "Object",
[[Properties]]: «
{
 [[Key]]: "myself",
 [[Value]]: <a pointer to this whole structure>
}
»
}
```

2.7.4 $\text{StructuredSerialize ( value )}^{p10}$

1. Return ? $\text{StructuredSerializeInternal}^{p105}(\text{value, false})$.

2.7.5 $\text{StructuredSerializeForStorage ( value )}^{p10}$

1. Return ? $\text{StructuredSerializeInternal}^{p105}(\text{value, true})$.

2.7.6 $\text{StructuredDeserialize ( serialized, targetRealm [ , memory ] )}^{p10}$

The $\text{StructuredDeserialize}^{p108}$ abstract operation takes as input a Record $\text{serialized}$, which was previously produced by $\text{StructuredSerialize}^{p108}$ or $\text{StructuredSerializeForStorage}^{p108}$, and deserializes it into a new JavaScript value, created in targetRealm.

This process can throw an exception, for example when trying to allocate memory for the new objects (especially ArrayBuffer objects).

1. If $\text{memory}$ was not supplied, let $\text{memory}$ be an empty map.

   Note

   The purpose of the memory map is to avoid deserializing objects twice. This ends up preserving cycles and the identity of duplicate objects in graphs.

2. If $\text{memory[serialized]}$ exists, then return $\text{memory[serialized]}$. 

108
3. Let \textit{deep} be false.

4. Let \textit{value} be an uninitialized value.

5. If \textit{serialized}.[[Type]] is "primitive", then set \textit{value} to \textit{serialized}.[[Value]].

6. Otherwise, if \textit{serialized}.[[Type]] is "Boolean", then set \textit{value} to a new Boolean object in \textit{targetRealm} whose [[BooleanData]] internal slot value is \textit{serialized}.[[BooleanData]].

7. Otherwise, if \textit{serialized}.[[Type]] is "Number", then set \textit{value} to a new Number object in \textit{targetRealm} whose [[NumberData]] internal slot value is \textit{serialized}.[[NumberData]].

8. Otherwise, if \textit{serialized}.[[Type]] is "BigInt", then set \textit{value} to a new BigInt object in \textit{targetRealm} whose [[BigIntData]] internal slot value is \textit{serialized}.[[BigIntData]].

9. Otherwise, if \textit{serialized}.[[Type]] is "String", then set \textit{value} to a new String object in \textit{targetRealm} whose [[StringData]] internal slot value is \textit{serialized}.[[StringData]].

10. Otherwise, if \textit{serialized}.[[Type]] is "Date", then set \textit{value} to a new Date object in \textit{targetRealm} whose [[DateValue]] internal slot value is \textit{serialized}.[[DateValue]].

11. Otherwise, if \textit{serialized}.[[Type]] is "RegExp", then set \textit{value} to a new RegExp object in \textit{targetRealm} whose [[RegExpMatcher]] internal slot value is \textit{serialized}.[[RegExpMatcher]], whose [[OriginalSource]] internal slot value is \textit{serialized}.[[OriginalSource]], and whose [[OriginalFlags]] internal slot value is \textit{serialized}.[[OriginalFlags]].

12. Otherwise, if \textit{serialized}.[[Type]] is "SharedArrayBuffer", then:
   1. If \textit{targetRealm}'s corresponding agent cluster is not \textit{serialized}.[[AgentCluster]], then then throw a "DataCloneError" DOMException.
   2. Otherwise, set \textit{value} to a new SharedArrayBuffer object in \textit{targetRealm} whose [[ArrayBufferData]] internal slot value is \textit{serialized}.[[ArrayBufferData]] and whose [[ArrayBufferByteLength]] internal slot value is \textit{serialized}.[[ArrayBufferByteLength]].

13. Otherwise, if \textit{serialized}.[[Type]] is "ArrayBuffer", then set \textit{value} to a new ArrayBuffer object in \textit{targetRealm} whose [[ArrayBufferData]] internal slot value is \textit{serialized}.[[ArrayBufferData]] and whose [[ArrayBufferByteLength]] internal slot value is \textit{serialized}.[[ArrayBufferByteLength]].

   If this throws an exception, catch it, and then throw a "DataCloneError" DOMException.

14. Otherwise, if \textit{serialized}.[[Type]] is "ArrayBufferView", then:
   1. Let \textit{deserializedArrayBuffer} be ? \texttt{StructuredDeserialize}\textsuperscript{\texttt{p108}}(\textit{serialized}.[[ArrayBufferSerialized]], \textit{targetRealm}, \textit{memory}).
   2. If \textit{serialized}.[[Constructor]] is "DataView", then set \textit{value} to a new DataView object in \textit{targetRealm} whose [[ViewedArrayBuffer]] internal slot value is \textit{deserializedArrayBuffer}, whose [[ByteLength]] internal slot value is \textit{serialized}.[[ByteLength]], and whose [[ByteOffset]] internal slot value is \textit{serialized}.[[ByteOffset]].
   3. Otherwise, set \textit{value} to a new typed array object in \textit{targetRealm}, using the constructor given by \textit{serialized}.[[Constructor]], whose [[ViewedArrayBuffer]] internal slot value is \textit{deserializedArrayBuffer}, whose [[TypedArrayName]] internal slot value is \textit{serialized}.[[Constructor]], whose [[ByteLength]] internal slot value is \textit{serialized}.[[ByteLength]], whose [[ByteOffset]] internal slot value is \textit{serialized}.[[ByteOffset]], and whose [[ArrayLength]] internal slot value is \textit{serialized}.[[ArrayLength]].

15. Otherwise, if \textit{serialized}.[[Type]] is "Map", then:
   1. Set \textit{value} to a new Map object in \textit{targetRealm} whose [[MapData]] internal slot value is a new empty \texttt{List}.
   2. Set \textit{deep} to true.

16. Otherwise, if \textit{serialized}.[[Type]] is "Set", then:
   1. Set \textit{value} to a new Set object in \textit{targetRealm} whose [[SetData]] internal slot value is a new empty \texttt{List}.
   2. Set \textit{deep} to true.
17. Otherwise, if `serialized.[[Type]]` is "Array", then:
   1. Let `outputProto` be `targetRealm.[[Intrinsics]].[%Array.prototype%].
   2. Set `value` to `! ArrayCreate(serialized.[[Length]], outputProto).`
   3. Set `deep` to true.

18. Otherwise, if `serialized.[[Type]]` is "Object", then:
   1. Set `value` to a new Object in `targetRealm`.
   2. Set `deep` to true.

19. Otherwise, if `serialized.[[Type]]` is "Error", then:
   1. Let `prototype` be `%Error.prototype%`.
   2. If `serialized.[[Name]]` is "EvalError", then set `prototype` to `%EvalError.prototype%`.
   3. If `serialized.[[Name]]` is "RangeError", then set `prototype` to `%RangeError.prototype%`.
   4. If `serialized.[[Name]]` is "ReferenceError", then set `prototype` to `%ReferenceError.prototype%`.
   5. If `serialized.[[Name]]` is "SyntaxError", then set `prototype` to `%SyntaxError.prototype%`.
   6. If `serialized.[[Name]]` is "TypeError", then set `prototype` to `%TypeError.prototype%`.
   7. If `serialized.[[Name]]` is "URIError", then set `prototype` to `%URIError.prototype%`.
   8. Let `message` be `serialized.[[Message]]`.
   9. Set `value` to `! ObjectCreate(prototype, « [[ErrorData]] »)`.
   10. Let `messageDesc` be `PropertyDescriptor( [[Value]]: message, [[Writable]]: true, [[Enumerable]]: false, [[Configurable]]: true )`.
   11. If `message` is not undefined, then perform `! OrdinaryDefineOwnProperty(value, "message", messageDesc)`.
   12. Any interesting accompanying data attached to `serialized` should be deserialized and attached to `value`.

20. Otherwise:
   1. Let `interfaceName` be `serialized.[[Type]]`.
   2. If the interface identified by `interfaceName` is not `exposed` in `targetRealm`, then throw a "DataCloneError" `DOMException`.
   3. Set `value` to a new instance of the interface identified by `interfaceName`, created in `targetRealm`.
   4. Set `deep` to true.


22. If `deep` is true, then:
   1. If `serialized.[[Type]]` is "Map", then:
      1. For each `Record { [[Key]], [[Value]] }` entry of `serialized.[[MapData]]`:
         1. Let `deserializedKey` be ? `StructuredDeserializer(entry.[[Key]], targetRealm, memory)`.
         2. Let `deserializedValue` be ? `StructuredDeserializer(entry.[[Value]], targetRealm, memory)`.
         3. Append { [[Key]]: `deserializedKey`, [[Value]]: `deserializedValue` } to `value.[[MapData]]`.
   2. Otherwise, if `serialized.[[Type]]` is "Set", then:
      1. For each `entry` of `serialized.[[SetData]]`:
         1. Let `deserializedEntry` be ? `StructuredDeserializer(entry, targetRealm, memory)`.
         2. Append `deserializedEntry` to `value.[[SetData]]`. 
3. Otherwise, if `serialized.[[Type]]` is "Array" or "Object", then:
   1. For each Record `{ [[Key]], [[Value]] }` entry of `serialized.[[Properties]]`:
      1. Let `deserializedValue` be `StructuredDeserialize`(`entry.[[Value]], targetRealm, memory`).
      2. Let `result` be `CreateDataProperty`(value, entry.[[Key]], `deserializedValue`).
      3. Assert: `result` is true.
   4. Otherwise:
      1. Perform the appropriate deserialization steps for the interface identified by `serialized.[[Type]]`, given `serialized` and `value`.
         The deserialization steps may need to perform a sub-deserialization. This is an operation which takes as input a previously-serialized Record `subSerialized`, and returns `StructuredDeserialize`(`subSerialized, targetRealm, memory`). (In other words, a sub-deserialization is a specialization of `StructuredDeserialize` to be consistent within this invocation.)

23. Return `value`.

2.7.7 `StructuredSerializeWithTransfer` (value, transferList) §p11

1. Let `memory` be an empty map.

   **Note**
   In addition to how it is used normally by `StructuredSerializeInternal`, in this algorithm memory is also used to ensure that `StructuredSerializeInternal` ignores items in transferList, and let us do our own handling instead.

2. For each `transferable` of `transferList`:
   1. If `transferable` has neither an `[[ArrayBufferData]]` internal slot nor a `[[Detached]]` internal slot, then throw a "DataCloneError" DOMException.
   2. If `transferable` has an `[[ArrayBufferData]]` internal slot and `IsSharedArrayBuffer`(`transferable`) is true, then throw a "DataCloneError" DOMException.
   3. If `memory[transferable]` exists, then throw a "DataCloneError" DOMException.
   4. Set `memory[transferable]` to { `[[Type]]`: an uninitialized value }.

   **Note**
   transferable is not transferred yet as transferring has side effects and `StructuredSerializeInternal` needs to be able to throw first.

3. Let `serialized` be `StructuredSerializeInternal`(`value`, false, `memory`).
4. Let `transferDataHolders` be a new empty List.
5. For each `transferable` of `transferList`:
   1. If `transferable` has an `[[ArrayBufferData]]` internal slot and `IsDetachedBuffer`(`transferable`) is true, then throw a "DataCloneError" DOMException.
   2. If `transferable` has a `[[Detached]]` internal slot and `transferable.[[Detached]]` is true, then throw a "DataCloneError" DOMException.
   3. Let `dataHolder` be `memory[transferable]`.
   4. If `transferable` has an `[[ArrayBufferData]]` internal slot, then:
      1. Set `dataHolder.[[Type]]` to "ArrayBuffer".
      2. Set `dataHolder.[[ArrayBufferData]]` to `transferable.[[ArrayBufferData]]`. 
3. Set `dataHolder.[[ArrayBufferByteLength]]` to `transferable.[[ArrayBufferByteLength]]`.

4. Perform `DetachArrayBuffer(transferable)`.

5. Otherwise:
   1. Assert: `transferable` is a `platform object` that is a `transferable object`.
   2. Let `interfaceName` be the identifier of the `primary interface` of `transferable`.
   3. Set `dataHolder.[[Type]]` to `interfaceName`.
   4. Perform the appropriate `transfer steps` for the interface identified by `interfaceName`, given `transferable` and `dataHolder`.
   5. Set `transferable.[[Detached]]` to true.

6. Append `dataHolder` to `transferDataHolders`.

6. Return `{ [[Serialized]]: serialized, [[TransferDataHolders]]: transferDataHolders }`.

### 2.7.8 StructuredDeserializeWithTransfer ( `serializeWithTransferResult`, `targetRealm` )

1. Let `memory` be an empty `map`.

   **Note**
   Analogous to `StructuredSerializeWithTransfer`, in addition to how it is used normally by `StructuredDeserialize`, in this algorithm `memory` is also used to ensure that `StructuredDeserialize` ignores items in `serializeWithTransferResult.[[TransferDataHolders]]`, and let us do our own handling instead.

2. Let `transferredValues` be a new empty `List`.

3. For each `transferDataHolder` of `serializeWithTransferResult.[[TransferDataHolders]]`:
   1. Let `value` be an uninitialized value.
   2. If `transferDataHolder.[[Type]]` is "ArrayBuffer", then set `value` to a new `ArrayBuffer` object in `targetRealm` whose `[[ArrayBufferData]]` internal slot value is `transferDataHolder.[[ArrayBufferData]]`, and whose `[[ArrayBufferByteLength]]` internal slot value is `transferDataHolder.[[ArrayBufferByteLength]]`.

   **Note**
   In cases where the original memory occupied by `[[ArrayBufferData]]` is accessible during the deserialization, this step is unlikely to throw an exception, as no new memory needs to be allocated: the memory occupied by `[[ArrayBufferData]]` is instead just getting transferred into the new `ArrayBuffer`. This could be true, for example, when both the source and target Realms are in the same process.

3. Otherwise:
   1. Let `interfaceName` be `transferDataHolder.[[Type]]`.
   2. If the interface identified by `interfaceName` is not exposed in `targetRealm`, then throw a "DataCloneError" `DOMException`.
   3. Set `value` to a new instance of the interface identified by `interfaceName`, created in `targetRealm`.
   4. Perform the appropriate `transfer-receiving steps` for the interface identified by `interfaceName` given `transferDataHolder` and `value`.
   5. Set `memory[transferDataHolder]` to `value`.

6. Append `value` to `transferredValues`.

Specifications can use the `[[ArrayBufferDetachKey]]` internal slot to prevent `ArrayBuffers` from being detached. This is used in WebAssembly JavaScript Interface, for example. [WASMJS]
4. Let `deserialized` be `StructuredDeserialize{serializeWithTransferResult([[Serialized]], targetRealm, memory)}`.

5. Return `{ [[Deserialized]]: `deserialized`, [[TransferredValues]]: `transferredValues` }`.

**2.7.9 Performing serialization and transferring from other specifications**

Other specifications may use the abstract operations defined here. The following provides some guidance on when each abstract operation is typically useful, with examples.

*StructuredSerializeWithTransfer*

*StructuredDeserializeWithTransfer*

Cloning a value to another JavaScript Realm, with a transfer list, but where the target Realm is not known ahead of time. In this case the serialization step can be performed immediately, with the deserialization step delayed until the target Realm becomes known.

Example

```javascript
messagePort.postMessage() uses this pair of abstract operations, as the destination Realm is not known until the MessagePort has been shipped.
```

*StructuredSerialize*

*StructuredSerializeForStorage*

*StructuredDeserialize*

Creating a JavaScript Realm-independent snapshot of a given value which can be saved for an indefinite amount of time, and then reified back into a JavaScript value later, possibly multiple times.

*StructuredSerializeForStorage* can be used for situations where the serialization is anticipated to be stored in a persistent manner, instead of passed between Realms. It throws when attempting to serialize `SharedArrayBuffer` objects, since storing shared memory does not make sense. Similarly, it can throw or possibly have different behavior when given a platform object with custom serialization steps when the `forStorage` argument is true.

Example

```javascript
history.pushState() and history.replaceState() use StructuredSerializeForStorage on author-supplied state objects, storing them as serialized state in the appropriate session history entry. Then, StructuredDeserialize is used so that the history.state property can return a clone of the originally-supplied state object.
```

Example

```javascript
broadcastChannel.postMessage() uses StructuredSerialize on its input, then uses StructuredDeserialize multiple times on the result to produce a fresh clone for each destination being broadcast to. Note that transferring does not make sense in multi-destination situations.
```

Example

Any API for persisting JavaScript values to the filesystem would also use StructuredSerializeForStorage on its input and StructuredDeserialize on its output.

In general, call sites may pass in Web IDL values instead of JavaScript values; this is to be understood to perform an implicit conversion to the JavaScript value before invoking these algorithms.

**Note**

This specification used to define a "structured clone" algorithm, and more recently a StructuredClone abstract operation. However, in practice all known uses of it were better served by separate serialization and deserialization steps, so it was removed.

Call sites that are not invoked as a result of author code synchronously calling into a user agent method must take care to properly prepare to run script and prepare to run a callback before invoking StructuredSerialize, StructuredSerializeForStorage, or StructuredSerializeWithTransfer abstract operations, if they are being performed on arbitrary objects. This is necessary because the serialization process can invoke author-defined accessors as part of its final deep-serialization steps, and these accessors could call into operations that rely on the entry and incumbent concepts being properly set up.
window.postMessage() performs StructuredSerializeWithTransfer on its arguments, but is careful to do so immediately, inside the synchronous portion of its algorithm. Thus it is able to use the algorithms without needing to prepare to run script and prepare to run a callback.

Example

In contrast, a hypothetical API that used StructuredSerialize to serialize some author-supplied object periodically, directly from a task on the event loop, would need to ensure it performs the appropriate preparations beforehand. As of this time, we know of no such APIs on the platform; usually it is simpler to perform the serialization ahead of time, as a synchronous consequence of author code.
3 Semantics, structure, and APIs of HTML documents

3.1 Documents

Every XML and HTML document in an HTML UA is represented by a Document object. The Document object's URL is defined in DOM. It is initially set when the Document object is created, but can change during the lifetime of the Document object; for example, it changes when the user navigates to a fragment on the page and when the pushState() method is called with a new URL.

⚠ Warning!
Interactive user agents typically expose the Document object's URL in their user interface. This is the primary mechanism by which a user can tell if a site is attempting to impersonate another.

The Document object's origin is defined in DOM. It is initially set when the Document object is created, and can change during the lifetime of the Document only upon setting document.domain. A Document's origin may differ from the origin of its URL; for example when a nested browsing context is created, its active Document's origin is set to the nested browsing context's container document's origin, even though its active Document's URL is "about:blank".

When a Document is created by a script using the createDocument() or createHTMLDocument() the Document is ready for post-load tasks, immediately.

The document's referrer is a string (representing a URL) that can be set when the Document is created. If it is not explicitly set, then its value is the empty string.

3.1.1 The Document object

DOM defines a Document interface, which this specification extends significantly.

```
enum DocumentReadyState { "loading", "interactive", "complete" };
typedef (HTMLScriptElement or SVGScriptElement) HTMLOrSVGScriptElement;

[LegacyOverrideBuiltIns]
partial interface Document {
    // resource metadata management
    [PutForwards="href", LegacyUnforgeable] readonly attribute Location? location;
    attribute USVString domain;
    readonly attribute USVString referrer;
    attribute USVString cookie;
    readonly attribute DOMString lastModified;
    readonly attribute DocumentReadyState readyState;

    // DOM tree accessors
    getter object (DOMString name);
    [CEReactions] attribute DOMString title;
    [CEReactions] attribute DOMString dir;
    [CEReactions] attribute HTMLCollection<HTMLCollection> images;
    [SameObject] readonly attribute HTMLCollection scripts;
    [SameObject] readonly attribute HTMLCollection forms;
    [SameObject] readonly attribute HTMLCollection links;
    [SameObject] readonly attribute HTMLCollection plugins;
    [SameObject] readonly attribute HTMLCollection embeds;
}
```

MDN 115
readonly attribute HTMLOrSVGScriptElement? currentScript; // classic scripts in a document tree only

// dynamic markup insertion
[CEReactions] Document open(optional DOMString unused1, optional DOMString unused2); // both arguments are ignored
WindowProxy? open(USVString url, DOMString name, DOMString features);
[CEReactions] undefined close();
[CEReactions] undefined write(DOMString... text);
[CEReactions] undefined writeln(DOMString... text);

// user interaction
readonly attribute WindowProxy? defaultView;
boolean hasFocus();
[CEReactions] attribute DOMString designMode;
[CEReactions] boolean execCommand(DOMString commandId, optional boolean showUI = false, optional DOMString value = "");
boolean queryCommandEnabled(DOMString commandId);
boolean queryCommandIndeterminate(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
[LegacyElementThis] attribute EventHandler onreadystatechange;

// also has obsolete members
};
Document includes GlobalEventHandlers;
Document includes DocumentAndElementEventHandlers;

The Document has a referrer policy (a referrer policy), initially the empty string, which represents the default referrer policy used by fetches initiated by the Document.

The Document has an embedder policy (an embedder policy).

The Document has a CSP list, which is a CSP list containing all of the Content Security Policy objects active for the document. The list is empty unless otherwise specified.

The Document has a permissions policy, which is a permissions policy, which is initially empty.

The Document has a module map, which is a module map, initially empty.

The Document has a cross-origin opener policy, which is a cross-origin opener policy.

3.1.2 The DocumentOrShadowRoot interface

DOM defines the DocumentOrShadowRoot mixin, which this specification extends.

```idl
partial interface mixin DocumentOrShadowRoot {
  readonly attribute Element? activeElement;
};
```

3.1.3 Resource metadata management

For web developers (non-normative)
The `referrer` attribute must return the document's referrer.

The `referrer` attribute represents the document's referrer, which is the URL 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 `referrer` link type can be used to block the referrer.

For web developers (non-normative)

The `cookie` attribute represents the cookies of the resource identified by the document's URL.

A Document object that falls into one of the following conditions is a cookie-averse Document object:

- A Document object whose browsing context is null.
- A Document whose URL's scheme is not an HTTP(S) scheme.

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 URL for a "non-HTTP" API, decoded using UTF-8 decode 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 URL 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.

For web developers (non-normative)

The `lastModified` attribute 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.

For web developers (non-normative)

```
document.readyState
```

Returns "loading" while the `Document` is loading, "interactive" once it is finished parsing but still loading subresources, and "complete" once it has loaded.

The `readystatechange` event fires on the `Document` object when this value changes.

The `DOMContentLoaded` event fires after the transition to "interactive" but before the transition to "complete", at the point where all subresources apart from `async` script elements have loaded.

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 an 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.

3.1.4 DOM tree accessors

The `html` element of a document is its document element, if it's an `html` element, and null otherwise.

For web developers (non-normative)

```
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).

For web developers (non-normative)

For web developers (non-normative)
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 document element is an **SVG** `svg` element, then let `value` be the child text content of the first `SVG title` element that is a child of the document element.
2. Otherwise, let `value` be the child text content of the `title` element, or the empty string if the `title` element is null.
3. Strip and collapse ASCII 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 document element is an **SVG** `svg` element**

1. If there is an `SVG title` element that is a child of the document element, let `element` be the first such element.
2. Otherwise:
   1. Let `element` be the result of creating an element given the document element's node document, `title`, and the SVG namespace.
   2. Insert `element` as the first child of the document element.
3. String replace all with the given value within `element`.

**If the document element is in the **HTML** namespace**

1. If the `title` element is null and the `head` element is null, then return.
2. If the `title` element is non-null, let `element` be the `title` element.
3. Otherwise:
   1. Let `element` be the result of creating an element given the document element's node document, `title`, and the HTML namespace.
   2. Append `element` to the `head` element.
4. String replace all with the given value within `element`.

Otherwise

Do nothing.

---

**For web developers (non-normative)**

`document . body` 

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" *DOMException*.

The **body element** of a document is the first of the `html` element's children that is either a `body` element or a `frameset` element, or null if there is no such element.

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" *DOMException*.
2. Otherwise, if the new value is the same as \texttt{the body element}©, return.

3. Otherwise, if \texttt{the body element}© is not null, then replace \texttt{the body element}© with the new value within \texttt{the body element}©’s parent and return.

4. Otherwise, if there is no \texttt{document element}, throw a "HierarchyRequestError" \texttt{DOMException}.

5. Otherwise, \texttt{the body element}© is null, but there’s a \texttt{document element}. Append the new value to the \texttt{document element}.

\textbf{Note}

The value returned by the \texttt{body}© getter is not always the one passed to the setter.

\textbf{Example}

In this example, the setter successfully inserts a \texttt{body}© element (though this is non-conforming since SVG does not allow a \texttt{body}© as child of \texttt{SVG svg}). However the getter will return null because the document element is not \texttt{html}©.

\begin{verbatim}
<svg xmlns="http://www.w3.org/2000/svg">
  <script>
    document.body = document.createElementNS("http://www.w3.org/1999/xhtml", "body");
    console.assert(document.body === null);
  </script>
</svg>
\end{verbatim}

\textbf{For web developers (non-normative)}

\texttt{document.images}©

Returns an \texttt{HTMLCollection} of the \texttt{img}© elements in the \texttt{Document}©.

\texttt{document.embeds}©

\texttt{document.plugins}©

Return an \texttt{HTMLCollection} of the \texttt{embed}© elements in the \texttt{Document}©.

\texttt{document.links}©

Returns an \texttt{HTMLCollection} of the \texttt{a}© and \texttt{area}© elements in the \texttt{Document}© that have \texttt{href}© attributes.

\texttt{document.forms}©

Return an \texttt{HTMLCollection} of the \texttt{form}© elements in the \texttt{Document}©.

\texttt{document.scripts}©

Return an \texttt{HTMLCollection} of the \texttt{script}© elements in the \texttt{Document}©.

The \texttt{images} attribute must return an \texttt{HTMLCollection} rooted at the \texttt{Document}© node, whose filter matches only \texttt{img}© elements.

The \texttt{embeds} attribute must return an \texttt{HTMLCollection} rooted at the \texttt{Document}© node, whose filter matches only \texttt{embed}© elements.

The \texttt{plugins} attribute must return the same object as that returned by the \texttt{embeds}© attribute.

The \texttt{links} attribute must return an \texttt{HTMLCollection} rooted at the \texttt{Document}© node, whose filter matches only \texttt{a}© elements with \texttt{href}© attributes and \texttt{area}© elements with \texttt{href}© attributes.

The \texttt{forms} attribute must return an \texttt{HTMLCollection} rooted at the \texttt{Document}© node, whose filter matches only \texttt{form}© elements.

The \texttt{scripts} attribute must return an \texttt{HTMLCollection} rooted at the \texttt{Document}© node, whose filter matches only \texttt{script}© elements.

\textbf{For web developers (non-normative)}

\texttt{collection = document.getElementsByTagName}©(\textit{name})

Returns a \texttt{NodeList} of elements in the \texttt{Document}© that have a \texttt{name} attribute with the value \textit{name}.

The \texttt{getElementsByTagName}©(\textit{name}) method steps are to return a \texttt{live}© \texttt{NodeList} containing all the \texttt{HTML elements}© in that document.
that have a name attribute whose value is identical to the name argument, 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.

For web developers (non-normative)

**document.currentScript**

Returns the script, element, or the SVG element, that is currently executing, as long as the element represents a classic script. 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 or SVG script element (e.g., because the running script is an event handler, or a timeout), or if the currently executing script or SVG script element represents a module script.

The currentScript attribute, on getting, must return the value to which it was most recently set. When the Document is created, the currentScript must be initialized to null.

Note

This API has fallen out of favor in the implementer and standards community, as it globally exposes script or SVG script elements. As such, it is not available in newer contexts, such as when running module scripts or when running scripts in a shadow tree. We are looking into creating a new solution for identifying the running script in such contexts, which does not make it globally available: see issue #1013.

The Document interface supports named properties. The supported property names of a Document object document at any moment consist of the following, in tree order according to the element that contributed them, ignoring later duplicates, and with values from id attributes coming before values from name attributes when the same element contributes both:

- the value of the name content attribute for all exposed embed, form, iframe, img, and exposed object elements that have a non-empty name content attribute and are in a document tree with document as their root;
- the value of the id content attribute for all exposed object elements that have a non-empty id content attribute and are in a document tree with document as their root; and
- the value of the id content attribute for all img elements that have both a non-empty id content attribute and a non-empty name content attribute, and are in a document tree with document as their root.

To determine the value of a named property name for a Document, the user agent must return the value obtained using the following steps:

1. Let elements be the list of named elements with the name that are in a document tree with the Document as their root.

   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, and that iframe element’s nested browsing context is not null, then return the WindowProxy object of the element’s nested browsing context.

3. Otherwise, if elements has only one element, return that element.

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:

- Exposed embed, form, iframe, img, or exposed object elements that have a name content attribute whose value is name, 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.

3.2 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, such as web browsers or search engines, to present and use documents and applications in a wide variety of contexts that the author might not have considered.

Example
As a simple example, consider a web page written by an author who only considered desktop computer web browsers:

```html
<!DOCTYPE HTML>
<html lang="en">
  <head>
    <title>My Page</title>
  </head>
  <body>
    <h1>Welcome to my page</h1>
    <p>I like cars and lorries and have a big Jeep!</p>
    <h2>Where I live</h2>
    <p>I live in a small hut on a mountain!</p>
  </body>
</html>
```

Because HTML conveys meaning, rather than presentation, the same page can also be used by a small browser on a mobile phone, without any change to the page. Instead of headings being in large letters as on the desktop, for example, the browser on the mobile phone might use the same size text for the whole page, but with the headings in bold.

But it goes further than just differences in screen size: the same page could equally be used by a blind user using a browser based around speech synthesis, which instead of displaying the page on a screen, reads the page to the user, e.g. using headphones. Instead of large text for the headings, the speech browser might use a different volume or a slower voice.

That’s not all, either. Since the browsers know which parts of the page are the headings, they can create a document outline that the user can use to quickly navigate around the document, using keys for “jump to next heading” or “jump to previous heading”. Such features are especially common with speech browsers, where users would otherwise find quickly navigating a page quite difficult.

Even beyond browsers, software can make use of this information. Search engines can use the headings to more effectively index a page, or to provide quick links to subsections of the page from their results. Tools can use the headings to create a table of contents (that is in fact how this very specification’s table of contents is generated).

This example has focused on headings, but the same principle applies to all of the semantics in HTML.

Authors must not use elements, attributes, or attribute values for purposes other than their appropriate intended semantic purpose, as doing so prevents software from correctly processing the page.

Example
For example, the following snippet, intended to represent the heading of a corporate site, is non-conforming because the second line is not intended to be a heading of a subsection, but merely a subheading or subtitle (a subordinate heading for the same
Authors must not use elements, attributes, or attribute values that are not permitted by this specification or other applicable specifications, as doing so makes it significantly harder for the language to be extended in the future.

Example

The document in this next example is similarly non-conforming, despite being syntactically correct, because the data placed in the cells is clearly not tabular data, and the cite element mis-used:

```
<!DOCTYPE HTML>
<html lang="en-GB">
<head>  
title> Demonstration </title>  
</head>
<body>
  
  <table>
    <tr>
      <td> My favourite animal is the cat. </td>
    </tr>
    <tr>
      <td> — <a href="https://example.org/~ernest/">Ernest</a>, in an essay from 1992 </td>
    </tr>
  </table>

</body>
</html>
```

This would make software that relies on these semantics fail: for example, a speech browser that allowed a blind user to navigate tables in the document would report the quote above as a table, confusing the user; similarly, a tool that extracted titles of works from pages would extract "Ernest" as the title of a work, even though it's actually a person's name, not a title.

A corrected version of this document might be:

```
<!DOCTYPE HTML>
<html lang="en-GB">
<head>  
title> Demonstration </title>  
</head>
<body>
  
  <blockquote>
    <p> My favourite animal is the cat. </p>
  </blockquote>
  
  <p> — <a href="https://example.org/~ernest/">Ernest</a>, in an essay from 1992 </p>

</body>
</html>
```
DOM nodes whose node document’s browsing context is null are exempt from all document conformance requirements other than the HTML syntax requirements and XML syntax requirements.

Example

In the next example, there is a non-conforming attribute value ("carpet") and a non-conforming attribute ("texture"), which is not permitted by this specification:

```html
<label>Carpet: <input type="carpet" name="c" texture="deep pile"></input></label>
```

Here would be an alternative and correct way to mark this up:

```html
<label>Carpet: <input type="text" class="carpet" name="c" data-texture="deep pile"></input></label>
```

Elements in the DOM represent things; that is, they have intrinsic meaning, also known as semantics.

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

For example, an `ol` element represents an ordered list.

Elements can be referenced (referred to) in some way, either explicitly or implicitly. One way that an element in the DOM can be explicitly referenced is by giving an `id` attribute to the element, and then creating a hyperlink with that `id` attribute's value as the fragment for the hyperlink's `href` attribute value. Hyperlinks are not necessary for a reference, however; any manner of referring to the element in question will suffice.
Consider the following `figure` element, which is given an `id` attribute:

```html
<figure id="module-script-graph">
  <img src="module-script-graph.svg" alt="Module A depends on module B, which depends on modules C and D."/>
  <figcaption>Figure 27: a simple module graph</figcaption>
</figure>
```

A `hyperlink`-based `reference` could be created using the `a` element, like so:

```html
As we can see in <a href="#module-script-graph">figure 27</a>, ...
```

However, there are many other ways of referencing the `figure` element, such as:

- "As depicted in the figure of modules A, B, C, and D..."
- "In Figure 27..." (without a hyperlink)
- "From the contents of the 'simple module graph' figure..."
- "In the figure below..." (but this is discouraged)

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.

```idl
[Exposed=Window]
interface HTMLElement : Element {
  [HTMLConstructor] constructor();

  // metadata attributes
  [CEReactions] attribute DOMString title;
  [CEReactions] attribute DOMString lang;
  [CEReactions] attribute boolean translate;
  [CEReactions] attribute DOMString dir;

  // user interaction
  [CEReactions] attribute boolean hidden;
  undefined click();
  [CEReactions] attribute DOMString accessKey;
  readonly attribute DOMString accessKeyLabel;
  [CEReactions] attribute boolean draggable;
  [CEReactions] attribute boolean spellcheck;
  [CEReactions] attribute DOMString autocapitalize;

  [CEReactions] attribute [LegacyNullToEmptyString] DOMString innerText;

  ElementInternals attachInternals();
};

HTMLElement includes GlobalEventHandlers;
HTMLElement includes DocumentAndElementEventHandlers;
HTMLElement includes ElementContentEditable;
HTMLElement includes HTMLOrSVGElement;

[Exposed=Window]
interface HTMLUnknownElement : HTMLElement {
  // Note: intentionally no [HTMLConstructor]
};
```
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 `element interface` for an element with name `name` in the `HTML namespace` is determined as follows:

1. If `name` is `applet`, `bgsound`, `blink`, `isindex`, `keygen`, `multicol`, `nextid`, or `spacer`, then return `HTMLUnknownElement`.
2. If `name` is `acronym`, `basefont`, `big`, `center`, `nobr`, `noembed`, `noframes`, `plaintext`, `rb`, `rtc`, `strike`, or `tt`, then return `HTMLElement`.
3. If `name` is `listing` or `xmp`, then return `HTMLPreElement`.
4. Otherwise, if this specification defines an interface appropriate for the `element type` corresponding to the local name `name`, then return that interface.
5. If other applicable specifications define an appropriate interface for `name`, then return the interface they define.
6. If `name` is a valid custom element name, then return `HTMLElement`.
7. Return `HTMLUnknownElement`.

Features shared between HTML and SVG elements use the `HTMLOrSVGElement` interface mixin:

```idl
interface mixin HTMLOrSVGElement {
  [SameObject] readonly attribute DOMStringMap dataset;
  attribute DOMString nonce; // intentionally no [CEReactions]

  [CEReactions] attribute boolean autofocus;
  [CEReactions] attribute long tabIndex;
  undefined focus(optional FocusOptions options = {});
  undefined blur();
};
```

An example of an element that is neither an HTML nor SVG element is one created as follows:

```javascript
const el = document.createElementNS("some namespace", "example");
console.assert(el.constructor === Element);
```

### 3.2.3 HTML element constructors

To support the `custom elements` feature, all HTML elements have special constructor behavior. This is indicated via the `[HTMLConstructor]` IDL extended attribute. It indicates that the interface object for the given interface will have a specific behavior when called, as defined in detail below.

The `[HTMLConstructor]` extended attribute must take no arguments, and must only appear on constructor operations. It must appear only once on a constructor operation, and the interface must contain only the single, annotated constructor operation, and no others. The annotated constructor operation must be declared to take no arguments.

Interfaces declared with constructor operations that are annotated with the `[HTMLConstructor]` extended attribute have the following overridden constructor steps:

1. Let `registry` be the `current global object`'s `CustomElementRegistry` object.
2. If `NewTarget` is equal to the `active function object`, then throw a `TypeError`.

Example

This can occur when a custom element is defined using an `element interface` as its constructor:

```javascript
customElements.define("bad-1", HTMLButtonElement);
new HTMLButtonElement(); // (1)
document.createElement("bad-1"); // (2)
```

In this case, during the execution of `HTMLButtonElement` (either explicitly, as in (1), or implicitly, as in (2)), both the `active function object` and `NewTarget` are `HTMLButtonElement`. If this check was not present, it would be possible to create an instance of `HTMLButtonElement` whose local name was `bad-1`.

3. Let `definition` be the entry in `registry` with `constructor` equal to `NewTarget`. If there is no such definition, then throw a `TypeError`.

Note

Since there can be no entry in registry with a `constructor` of undefined, this step also prevents HTML element constructors from being called as functions (since in that case `NewTarget` will be undefined).

4. Let `is value` be null.

5. If `definition's local name` is equal to `definition's name` (i.e., `definition` is for an `autonomous custom element`), then:

   1. If the `active function object` is not `HTMLElement`, then throw a `TypeError`.

Example

This can occur when a custom element is defined to not extend any local names, but inherits from a non-`HTMLElement` class:

```javascript
customElements.define("bad-2", class Bad2 extends HTMLParagraphElement {});
```

In this case, during the (implicit) `super()` call that occurs when constructing an instance of `Bad2`, the `active function object` is `HTMLParagraphElement`, not `HTMLElement`.

6. Otherwise (i.e., if `definition` is for a `customized built-in element`):

   1. Let `valid local names` be the list of local names for elements defined in this specification or in `other applicable specifications` that use the `active function object` as their `element interface`.

   2. If `valid local names` does not contain `definition's local name`, then throw a `TypeError`.

Example

This can occur when a custom element is defined to extend a given local name but inherits from the wrong class:

```javascript
customElements.define("bad-3", class Bad3 extends HTMLQuoteElement {}, { extends: "p" });
```

In this case, during the (implicit) `super()` call that occurs when constructing an instance of `Bad3`, `valid local names` is the list containing `g` and `blockquote`, but `definition's local name` is `p`, which is not in that list.

3. Set `is value` to `definition's name`.

7. If `definition's construction stack` is empty, then:

   1. Let `element` be the result of `internally creating a new object implementing the interface` to which the `active function object` corresponds, given the `current Realm Record` and `NewTarget`.

   2. Set `element's node document` to the `current global object`'s associated `Document`.

   3. Set `element's namespace` to the `HTML namespace`.
4. Set element's namespace prefix to null.
5. Set element's local name to definition's local name.
6. Set element's custom element state to "custom".
7. Set element's custom element definition to definition.
8. Set element's is value to is value.

Note
This occurs when author script constructs a new custom element directly, e.g. via new MyCustomElement().

8. Let prototype be Get(NewTarget, "prototype"). Rethrow any exceptions.
9. If Type(prototype) is not Object, then:
   1. Let realm be GetFunctionRealm(NewTarget).
   2. Set prototype to the interface prototype object of realm whose interface is the same as the interface of the active function object.

Note
The realm of the active function object might not be realm, so we are using the more general concept of "the same interface" across realms; we are not looking for equality of interface objects. This fallback behavior, including using the realm of NewTarget and looking up the appropriate prototype there, is designed to match analogous behavior for the JavaScript built-ins and Web IDL's internally create a new object implementing the interface algorithm.

10. Let element be the last entry in definition's construction stack.
11. If element is an already constructed marker, then throw an "InvalidStateError" DOMException.

Example
This can occur when the author code inside the custom element constructor non-conformantly creates another instance of the class being constructed, before calling super():

```javascript
let doSillyThing = false;

class DontDoThis extends HTMLElement {
  constructor() {
    if (doSillyThing) {
      doSillyThing = false;
      new DontDoThis();
      // Now the construction stack will contain an already constructed marker.
    }

    // This will then fail with an "InvalidStateError" DOMException:
    super();
  }
}
```

Example
This can also occur when author code inside the custom element constructor non-conformantly calls super() twice, since per the JavaScript specification, this actually executes the superclass constructor (i.e. this algorithm) twice, before throwing an error:

```javascript
class DontDoThisEither extends HTMLElement {
  constructor() {
    super();
    // This will throw, but not until it has already called into the HTMLElement
```

13. Replace the last entry in `definition's` construction stack with an already constructed marker.


Note

This step is normally reached when upgrading a custom element; the existing element is returned, so that the `super()` call inside the custom element constructor assigns that existing element to `this`.

In addition to the constructor behavior implied by `[HTMLConstructor]`, some elements also have named constructors (which are really factory functions with a modified prototype property).

Example

Named constructors for HTML elements can also be used in an extends clause when defining a custom element constructor:

```javascript
class AutoEmbiggenedImage extends Image {
    constructor(width, height) {
        super(width * 10, height * 10);
    }
}

customElements.define("auto-embiggened", AutoEmbiggenedImage, { extends: "img" });

const image = new AutoEmbiggenedImage(15, 20);
console.assert(image.width === 150);
console.assert(image.height === 200);
```

3.2.4 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, all phrasing content is flow content. Thus, elements that are phrasing content will only be listed as “where phrasing content is expected”, since this is the more-specific expectation. Anywhere that expects flow content also expects phrasing content, and thus also meets this expectation.

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.)

Accessibility considerations
For authors: Conformance requirements for use of ARIA role and aria-* attributes are defined in ARIA in HTML. [ARIA] [ARIAHTML]

For implementers: User agent requirements for implementing accessibility API semantics are defined in HTML Accessibility API Mappings. [HTMLAAM]

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.

3.2.4.1 Attributes
An attribute value is a string. Except where otherwise specified, attribute values on HTML elements may be any string value, including the empty string, and there is no restriction on what text can be specified in such attribute values.

3.2.5 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.

ASCII whitespace is 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
The Atom Syndication Format defines a content element. When its type attribute has the value xhtml, The Atom Syndication Format 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. [ATOM]

In addition, HTML elements may be orphan nodes (i.e. without a parent node).
3.2.5.1 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.*

3.2.5.2 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:

- Metadata content
- Flow content
- Sectioning content
- Heading content
- Phrasing content
- Embedded content
- 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.

3.2.5.2.1 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\textsuperscript{156}, link\textsuperscript{158}, meta\textsuperscript{165}, noscript\textsuperscript{167}, script\textsuperscript{161}, style\textsuperscript{176}, template\textsuperscript{169}, title\textsuperscript{155}

Elements from other namespaces whose semantics are primarily metadata-related (e.g. RDF) are also metadata\textsuperscript{132}.

Example

Thus, in the XML serialization, one can use RDF, like this:

```xml
<html xmlns="http://www.w3.org/1999/xhtml"
     xmlns:r="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xml:lang="en">
  <head>
    <title>Hedral's Home Page</title>
    <r: RDF>
      <Person xmlns="http://www.w3.org/2000/10/slap/pim/contact#"
               r:about="https://hedral.example.com/#">
        <fullName>Cat Hedral</fullName>
        <mailbox r:resource="mailto:hedral@damowmow.com/>
        <personalTitle>Sir</personalTitle>
      </Person>
    </r: RDF>
  </head>
  <body>
    <h1>My home page</h1>
    <p>I like playing with strings, I guess. Sister says squirrels are fun too so sometimes I follow her to play with them.</p>
  </body>
</html>
```

This isn't possible in the HTML serialization, however.

3.2.5.2.2 Flow content

Most elements that are used in the body of documents and applications are categorized as flow content.

- a\textsuperscript{229}, abbr\textsuperscript{251}, address\textsuperscript{219}, area\textsuperscript{444} (if it is a descendant of a map\textsuperscript{442} element), article\textsuperscript{181}, aside\textsuperscript{189}, audio\textsuperscript{384}, b\textsuperscript{274}, bdi\textsuperscript{277}, bdo\textsuperscript{268}, blockquote\textsuperscript{212}, br\textsuperscript{281}, button\textsuperscript{335}, canvas\textsuperscript{364}, cite\textsuperscript{247}, code\textsuperscript{326}, data\textsuperscript{268}, datalist\textsuperscript{254}, del\textsuperscript{313}, details\textsuperscript{2684}, dfn\textsuperscript{259}, dialog\textsuperscript{261}, div\textsuperscript{223}, dl\textsuperscript{228}, em\textsuperscript{242}, embed\textsuperscript{306}, fieldset\textsuperscript{562}, figure\textsuperscript{233}, footer\textsuperscript{216}, form\textsuperscript{468}, h1\textsuperscript{240}, h2\textsuperscript{191}, h3\textsuperscript{191}, h4\textsuperscript{191}, h5\textsuperscript{191}, h6\textsuperscript{191}, header\textsuperscript{215}, hgroup\textsuperscript{193}, hr\textsuperscript{215}, i\textsuperscript{273}, iframe\textsuperscript{362}, img\textsuperscript{321}, input\textsuperscript{499}, ins\textsuperscript{312}, kbd\textsuperscript{271}, label\textsuperscript{408}, link\textsuperscript{208} (if it is allowed in the body\textsuperscript{208}), main\textsuperscript{237} (if it is a hierarchically correct main element\textsuperscript{237}), map\textsuperscript{442}, mark\textsuperscript{276}, MathML, math, menu\textsuperscript{224}, meta\textsuperscript{152} (if the itemprop\textsuperscript{138} attribute is present), meter\textsuperscript{357}, nav\textsuperscript{218}, noscript\textsuperscript{263}, object\textsuperscript{273}, ol\textsuperscript{222}, output\textsuperscript{555}, p\textsuperscript{213}, picture\textsuperscript{317}, pre\textsuperscript{227}, progress\textsuperscript{535}, q\textsuperscript{248}, ruby\textsuperscript{253}, s\textsuperscript{246}, samp\textsuperscript{278}, script\textsuperscript{363}, section\textsuperscript{233}, select\textsuperscript{257}, small\textsuperscript{432}, small\textsuperscript{245}, span\textsuperscript{288}, strong\textsuperscript{242}, sub\textsuperscript{277}, sup\textsuperscript{272}, SVG, svg, table\textsuperscript{369}, template\textsuperscript{169}, textarea\textsuperscript{368}, time\textsuperscript{261}, u\textsuperscript{276}, ul\textsuperscript{233}, var\textsuperscript{269}, video\textsuperscript{380}, wbr\textsuperscript{282}, autonomous custom elements\textsuperscript{204}, text\textsuperscript{213}

3.2.5.2.3 Sectioning content

Sectioning content is content that defines the scope of headings\textsuperscript{213} and footers\textsuperscript{216}.

- article\textsuperscript{218}, aside\textsuperscript{218}, nav\textsuperscript{218}, section\textsuperscript{218}
Each sectioning content element potentially has a heading and an outline. See the section on 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.

3.2.5.2.4 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, hgroup

3.2.5.2.5 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, canvas, cite, code, data, datalist, del, dfn, em, embed, iframe, img, input, ins, kbd, label, link, span, strong, sub, sup, svg, template, textarea, time, u, var, video, wbr, autonomous custom elements, 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 ASCII whitespace).

Text nodes and attribute values must consist of scalar values, excluding noncharacters, and controls other than ASCII whitespace. This specification includes extra constraints on the exact value of Text nodes and attribute values depending on their precise context.

3.2.5.2.6 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, img, MathML, 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 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.

3.2.5.2.7 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, embed, iframe, img (if the usemap attribute is present), input (if the type attribute is not in the...
3.2.5.2.8 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 img 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, bdo, blockquote, button, canvas, cite, code, data, details, dfn, div, dl, dt, dd, figcaption, figure, footer, form, h1, h2, h3, h4, h5, h6, header, hgroup, i, iframe, img, input, ins, kbd, label, legend, li, link, main, map, mark, MathML, math, menu, ol, p, pre, progress, q, rt, s, samp, section, select, small, span, strong, sub, sup, svg, table, textarea, time, ul, var.

3.2.5.2.9 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.

3.2.5.3 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

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.
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.

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.

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.

Generally, having elements straddle paragraph boundaries is best avoided. Maintaining such markup can be difficult.

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.

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 MathML).

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**

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.

```html
<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:

```html
<header>
  <p>Welcome! <a href="about.html">This is home of...</a></p>
  <h1>The Falcons!</h1>
  <p>The Lockheed Martin multirole jet fighter aircraft!</p>
  This page discusses the F-16 Fighting Falcon's innermost secrets.
</header>
```

**Example**

It is possible for paragraphs to overlap when using certain elements that define fallback content. For example, in the following section:

```html
<section>
  <h1>My Cats</h1>
  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 "Use online simulator"
4. 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:

```html
<section>
  <h1>My Cats</h1>
  <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>
      <li><a href="https://sims.example.com/watch?v=LYds5xY4INU">Use online simulator</a></li>
    </ul>
    <p>Alternatively, upgrade to the Mellblom Browser.</p>
  </object>
  <p>I'm quite proud of it.</p>
</section>
```

### 3.2.6 Global attributes

The following attributes are common to and may be specified on all HTML elements (even those not defined in this specification):

- `accesskey`
- `autocapitalize`
- `autofocus`
- `contenteditable`
- `dir`
- `draggable`
- `enterkeyhint`
- `hidden`
- `inputmode`
- `is`
- `itemid`
- `itemprop`
- `itemref`
- `itemscope`
- `itemtype`
- `lang`
- `nonce`
- `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.
DOM defines the user agent requirements for the **class**, **id**, and **slot** attributes for any element in any namespace. [DOM]

The **class**, **id**, and **slot** attributes may be specified on all **HTML elements**.

When specified on **HTML elements**, the **class** attribute must have a value that is a set of space-separated tokens representing the various classes that the element belongs to.

**Note**

Assigning classes to an element affects class matching in selectors in CSS, the `getElementsByClassName()` method in the DOM, and other such features.

**Note**

The **id** attribute specifies its element's unique identifier (ID).

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.

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 fragments, 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.

There are no conformance requirements for the **slot** attribute specific to **HTML elements**.

**Note**

The **slot** attribute is used to assign a slot to an element: an element with a **slot** attribute is assigned to the slot created by the **slot** element whose name attribute's value matches that **slot** attribute's value — but only if that **slot** element finds itself in the shadow tree whose root's host has the corresponding **slot** attribute value.

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).

The following event handler content attributes may be specified on any **HTML element**:

- `onabort`
- `onauxclick`
- `onblur`
- `oncancel`
- `oncanplay`
Custom data attributes (e.g. data-foldername or data-msgid) can be specified on any HTML element to store custom data.
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.

In HTML, the xmlns attribute has absolutely no effect. It is basically a talisman. It is allowed merely to make migration to and from XML mildly easier. When parsed by an HTML parser\(^\text{p1072}\), the attribute ends up in no namespace, not the "http://www.w3.org/2000/xmlns/" namespace like namespace declaration attributes in XML do.

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.

XML 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\(^\text{p44}\), as the default behavior in HTML is to preserve whitespace. [XML]\(^\text{p1278}\)

There is no way to serialize the xml:space attribute on HTML elements\(^\text{p44}\) in the text/html\(^\text{p1236}\) syntax.

### 3.2.6.1 The title\(^{\text{p140}}\) attribute \(^{\text{p14}}\)

The title attribute represents\(^{\text{p124}}\) 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\(^{\text{p133}}\), it could be a label for, or instructions for, use of the element; and so forth. The value is text.

Relying on the title\(^{\text{p140}}\) 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\(^{\text{p140}}\) attribute of the nearest ancestor HTML element\(^{\text{p44}}\) with a title\(^{\text{p140}}\) 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\(^{\text{p140}}\) 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.

Caution is advised with respect to the use of newlines in title\(^{\text{p140}}\) 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\(^{\text{p138}}\), abbr\(^{\text{p243}}\), and input\(^{\text{p493}}\), define additional semantics for the title\(^{\text{p140}}\) 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 has a title\(^{\text{p140}}\) attribute, then return its value.
2. If the element has a parent element, then return the parent element's advisory information.

3. 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.

3.2.6.2 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 XML, 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.

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 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 node's inclusive ancestors do not 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.

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

Thus, for instance, an element with lang="xyzzy" would be matched by the selector :lang(xyzzy) (e.g. in CSS), but it would not be matched by :lang(abcd), 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 "xyzzy", 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 "xyzzy" was a
way to denote the Belarusian language, it would be *incorrect* for the screen reader to then start treating text as Belarusian, because "xyzzy" 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.

### 3.2.6.3 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 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 document 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 the `p` element
- `alt` on `area` and `img`, and `input` elements
- `content` on `meta` elements, if the `name` attribute specifies a metadata name whose value is known to be translatable.
- `download` on an `a` and `area` elements
- `label` on `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
- `srcref` 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` or the `Reset Button` state

Other specifications may define other attributes that are also translatable attributes. For example, ARIA would define the `aria-label` attribute as translatable.

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**

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:

```
```
3.2.6.4 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]\(^\text{[BIDI]}\)

**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:

1. If the element's `dir` attribute is in the `ltr` state
2. If the element is a `document element` and the `dir` attribute is not in a defined state (i.e. it is not present or has an invalid value)
3. If the element is a `input` 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)
4. If the element's `dir` attribute is in the `rtl` state
5. If the element is an `input` whose `type` attribute is in the `Text`, `Search`, `Telephone`, `URL`, or `Email` state, and the `dir` attribute is in the `auto` state
6. 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 \texttt{value}, then the directionality of the element is ‘\texttt{rtl}’. Otherwise, if the element’s \texttt{value} is not the empty string, or if the element is a \texttt{document element}, the directionality of the element is ‘\texttt{ltr}’.

Otherwise, the directionality of the element is the same as the element’s parent element’s directionality.

\begin{itemize}
  \item If the element’s \texttt{dir} attribute is in the \texttt{auto} state
  \item If the element is a \texttt{bdi} element and the \texttt{dir} attribute is not in a defined state (i.e. it is not present or has an invalid value)
\end{itemize}

Find the first character in tree order that matches the following criteria:

\begin{itemize}
  \item The character is from a \texttt{Text} node that is a descendant of the element whose directionality is being determined.
  \item The character is of bidirectional character type L, AL, or R.
  \item The character is not in a \texttt{Text} node that has an ancestor element that is a descendant of the element whose directionality is being determined and that is either:
    \begin{itemize}
      \item A \texttt{bdi} element.
      \item A \texttt{script} element.
      \item A \texttt{style} element.
      \item A \texttt{textarea} element.
      \item An element with a \texttt{dir} attribute in a defined state.
    \end{itemize}
\end{itemize}

If such a character is found and it is of bidirectional character type AL or R, the directionality of the element is ‘\texttt{rtl}’.

If such a character is found and it is of bidirectional character type L, the directionality of the element is ‘\texttt{ltr}’.

Otherwise, if the element is a \texttt{document element}, the directionality of the element is ‘\texttt{ltr}’.

Otherwise, the directionality of the element is the same as the element’s parent element’s directionality.

\begin{itemize}
  \item If the element has a parent element and the \texttt{dir} attribute is not in a defined state (i.e. it is not present or has an invalid value)
\end{itemize}

The directionality of the element is the same as the element’s parent element’s directionality.

Since the \texttt{dir} attribute is only defined for \texttt{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 ‘\texttt{ltr}’.

This attribute has rendering requirements involving the bidirectional algorithm.
• label on optgroup, option, and track elements
• placeholder on input and textarea elements
• title on all HTML elements

For web developers (non-normative)

```javascript
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

This markup fragment is of an IM conversation.

```
<p dir="auto" class="u1">Student</p>: How do you write "What's your name?" in Arabic?

<p dir="auto" class="u2">Teacher</p>: What's your name?

<p dir="auto" class="u1">Student</p>: My name is [insert name here].

<p dir="auto" class="u2">Teacher</p>: Thanks.

<p dir="auto" class="u1">Student</p>: Do you know how to write "Please"?

<p dir="auto" class="u2">Teacher</p>: Yes, it's [insert text here].
```

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?

Student: Thanks.

Teacher: That's written "[insert text here]."

Teacher: Do you know how to write "Please"?

Student: Yes, it's [insert text here].
```

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.

3.2.6.5 The `style` attribute

All HTML elements may have the `style` content attribute set. This is a `style attribute` as defined by CSS Style Attributes.

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 `style attributes`.

However, if the `Should element's inline behavior be blocked by Content Security Policy?` algorithm returns "Blocked" when executed upon the attribute's element, "style attribute", and the attribute's value, then the style rules defined in the attribute's value must...
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.)

The `style` IDL attribute is defined in CSS Object Model. [CSSOM]

**Example**

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.

```html
<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>
```

### 3.2.6.6 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 ASCII upper alphas.

**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, state, annotations, and similar, 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**

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.

```html
<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.
Every HTML element may have any number of custom data attributes specified, with any value. 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.

In this example, custom data attributes are used to store the result of a feature detection for PaymentRequest, which could be used in CSS to style a checkout page differently.

```html
<script>
  if ('PaymentRequest' in window) {
    document.documentElement.dataset.hasPaymentRequest = '';
  }
</script>
```

Here, the data-has-payment-request attribute is effectively being used as a boolean attribute; it is enough to check the presence of the attribute. However, if the author so wishes, it could later be populated with some value, maybe to indicate limited functionality of the feature.

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).

Every HTML element may have any number of custom data attributes specified, with any value. 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.

For web developers (non-normative)

```javascript
element.dataset
```

Returns a DOMStringMap object for the element's data-* attributes. Hyphenated names become camel-cased. For example, data-foo-bar=b" 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 whose associated element is this element.

The DOMStringMap interface is used for the dataset attribute. Each DOMStringMap has an associated element.

```idl
[Exposed=Window, LegacyOverrideBuiltIns]
interface DOMStringMap {
  getter DOMString (DOMString name);
  [CFReactions] setter undefined (DOMString name, DOMString value);
}
To get a DOMStringMap's name-value pairs, run the following algorithm:

1. Let list be an empty list of name-value pairs.
2. For each content attribute on the DOMStringMap's associated element whose first five characters are the string "data-" and whose remaining characters (if any) do not include any ASCII upper alphas, 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 an ASCII lower alpha, 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 supported property names on a DOMStringMap object at any instant are the names of each pair returned from getting the DOMStringMap's name-value pairs at that instant, in the order returned.

To determine the value of a named property name for a DOMStringMap, return the value component of the name-value pair whose name component is name in the list returned from getting the DOMStringMap's name-value pairs.

To set the value of a new named property or set the value of an existing named property for a DOMStringMap, given a property name and a new value, run the following steps:

1. If name contains a U+002D HYPHEN-MINUS character (-) followed by an ASCII lower alpha, then throw a "SyntaxError" DOMException.
2. For each ASCII upper alpha 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. If name does not match the XML Name production, throw an "InvalidCharacterError" DOMException.
5. Set an attribute value for the DOMStringMap's associated element using name and value.

To delete an existing named property name for a DOMStringMap, run the following steps:

1. For each ASCII upper alpha in name, insert a U+002D HYPHEN-MINUS character (-) before the character and replace the character with the same character converted to ASCII lowercase.
2. Insert the string data- at the front of name.
3. Remove an attribute by name given name and the DOMStringMap's associated element.

This algorithm will only get invoked by Web IDL for names that are given by the earlier algorithm for getting the DOMStringMap's name-value pairs.

Example

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="92432"
   data-weapons="laser 2" data-shields="50%"
   data-x="30" data-y="10" data-z="90">
   <button class="fire"
      onclick="spaceships[this.parentNode.dataset.shipId].fire()">Fire
</button>
```
Notice how the hyphenated attribute name becomes camel-cased in the API.

Example

Given the following fragment and elements with similar constructions:

```html
<img class="tower" id="tower5" data-x="12" data-y="5" data-ai="robotarget" data-hp="46" data-ability="flames" src="towers/rocket.png" alt="Rocket Tower">
```

...one could imagine a function `splashDamage()` that takes some arguments, the first of which is the element to process:

```javascript
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
    }
}
```

3.2.7 The `innerText` getter and setter

For web developers (non-normative)

```
element.innerText [ = value ]
```

Returns the element's text content "as rendered".

Can be set, to replace the element's children with the given value, but with line breaks converted to `br` elements.

The `innerText` getter steps are:

1. If this is not `being rendered` or if the user agent is a non-CSS user agent, then return this's descendant text content.

   **Note**
   
   This step can produce surprising results, as when the `innerText` getter is invoked on an element not `being rendered`, its text contents are returned, but when accessed on an element that is `being rendered`, all of its children that are not `being rendered` have their text contents ignored.

2. Let `results` be a new empty list.

3. For each child node `node` of this:

   1. Let `current` be the list resulting in running the `innerText collection steps` with `node`. Each item in `results` will either be a `string` or a positive integer (a required line break count).

   **Note**
   
   Intuitively, a required line break count item means that a certain number of line breaks appear at that point, but they can be collapsed with the line breaks induced by adjacent required line break count items, reminiscent to CSS margin-collapsing.
2. For each item item in current, append item to results.

4. Remove any items from results that are the empty string.

5. Replace any runs of consecutive required line break count items at the start or end of results.

6. Each remaining run of consecutive required line break count items with a string consisting of as many U+000A LINE FEED (LF) characters as the maximum of the values in the required line break count items.

7. Return the concatenation of the string items in results.

The inner text collection steps, given a node node, are as follows:

1. Let items be the result of running the inner text collection steps with each child node of node in tree order, and then concatenating the results to a single list.

2. If node's computed value of 'visibility' is not 'visible', then return items.

3. If node is not being rendered, then return items. For the purpose of this step, the following elements must act as described if the computed value of the 'display' property is not 'none':
   - select elements have an associated non-replaced inline CSS box whose child boxes include only those of optgroup and element child nodes;
   - optgroup elements have an associated non-replaced block-level CSS box whose child boxes include only those of element child nodes; and
   - option element have an associated non-replaced block-level CSS box whose child boxes are as normal for non-replaced block-level CSS boxes.

Note: items can be non-empty due to 'display:contents'.

4. If node is a Text node, then for each CSS text box produced by node, in content order, compute the text of the box after application of the CSS 'white-space' processing rules and 'text-transform' rules, set items to the list of the resulting strings, and return items. The CSS white-space processing rules are slightly modified: collapsible spaces at the end of lines are always collapsed, but they are only removed if the line is the last line of the block, or it ends with a br element. Soft hyphens should be preserved. [CSSTEXT]

5. If node is a br element, then append a string containing a single U+000A LINE FEED (LF) character to items.

6. If node's computed value of 'display' is 'table-cell', and node's CSS box is not the last 'table-cell' box of its enclosing 'table-row' box, then append a string containing a single U+0009 CHARACTER TABULATION (tab) character to items.

7. If node's computed value of 'display' is 'table-row', and node's CSS box is not the last 'table-row' box of the nearest ancestor 'table' box, then append a string containing a single U+000A LINE FEED (LF) character to items.

8. If node is a p element, then append 2 (a required line break count) at the beginning and end of items.

9. If node's used value of 'display' is block-level or 'table-caption', then append 1 (a required line break count) at the beginning and end of items. [CSSDISPLAY]

10. Return items.

Note: Floats and absolutely-positioned elements fall into this category.

This algorithm is amenable to being generalized to work on ranges. Then we can use it as the basis for Selection's stringifier and maybe expose it directly on ranges. See Bugzilla bug 10583.

The innerText setter steps are:

1. Let document be this's node document.
2. Let fragment be a new DocumentFragment object whose node document is document.

3. Let input be the given value.

4. Let position be a pointer into input, initially pointing at the start of the string.

5. Let text be the empty string.

6. While position is not past the end of input:
   1. Collect a sequence of code points that are not U+000A LINE FEED (LF) or U+000D CARRIAGE RETURN (CR) characters from input given position. Set text to the collected characters.
   2. If text is not the empty string, then append a new Text node whose data is text and node document is document to fragment.
   3. While position is not past the end of input, and the character at position is either a U+000A LINE FEED (LF) or U+000D CARRIAGE RETURN (CR) character:
      1. If the character at position is a U+000D CARRIAGE RETURN (CR) character and the next character is a U+000A LINE FEED (LF) character, then advance position to the next character in input.
      2. Advance position to the next character in input.
      3. Append the result of creating an element given document, br, and the HTML namespace to fragment.

7. Replace all with fragment within this.

### 3.2.8 Requirements relating to the bidirectional algorithm

#### 3.2.8.1 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). 

**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.

#### 3.2.8.2 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.

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 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.

The following elements and attributes have requirements defined by the 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
- bdo element
- bdi element
- br element
- pre element
- textarea element
- wbr element
3.2.9 Requirements related to ARIA and to platform accessibility APIs

User agent requirements for implementing Accessibility API semantics on HTML elements are defined in HTML Accessibility API Mappings. In addition to the rules there, for a custom element element, the default ARIA role semantics are determined as follows:

1. Let map be element's native accessibility semantics map.
2. If map["role"] exists, then return it.
3. Return no role.

Similarly, for a custom element element, the default ARIA state and property semantics, for a state or property named stateOrProperty, are determined as follows:

1. Let map be element's native accessibility semantics map.
2. If map[stateOrProperty] exists, then return it.
3. Return the default value for stateOrProperty.

**Note**
The "default semantics" referred to here are sometimes also called "native", "implicit", or "host language" semantics in ARIA. [ARIA]

**Note**
One implication of these definitions is that the default semantics can change over time. This allows custom elements the same expressivity as built-in elements; e.g., compare to how the default ARIA role semantics of an a element change as the href attribute is added or removed.

For an example of this in action, see the custom elements section.

Conformance checker requirements for checking use of ARIA role and aria-* attributes on HTML elements are defined in ARIA in HTML. [ARIAHTML]
4 The elements of HTML

4.1 The document element

4.1.1 The html element

Categories:
None.

Contexts in which this element can be used:
As document's document element.
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

Accessibility considerations:
For authors.
For implementers.

DOM interface:

```idl
[Exposed=Window]
interface HTMLHtmlElement : HTMLElement {
  [HTMLConstructor] constructor();

  // also has obsolete members
};
```

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.

Example

The html element in the following example declares that the document's language is English.

```html
<!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 can 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 can be omitted if the head element is not immediately followed by ASCII whitespace or a comment.

Content attributes:
Global attributes

Accessibility considerations:
For authors.
For implementers.

DOM interface:

```idl
[Exposed=Window]
interface HTMLHeadElement : HTMLElement {
    [HTMLConstructor] constructor();
};
```

The head element represents a collection of metadata for the Document.

Example

The collection of metadata in a head element can be large or small. Here is an example of a very short one:

```html
<!doctype html>
<html lang=en>
<head>
    <title>A document with a short head</title>
</head>
<body>
    ...
</body>
</html>
```

Here is an example of a longer one:

```html
<!DOCTYPE HTML>
<HTML LANG="EN">
<HEAD>
    <META CHARSET="UTF-8">
    <BASE HREF="https://www.example.com/">
    <TITLE>An application with a long head</TITLE>
    <LINK REL="STYLE SHEET" HREF="default.css">
    <LINK REL="STYLE SHEET ALTERNATE" HREF="big.css" TITLE="Big Text">
    <SCRIPT SRC="support.js"></SCRIPT>
    <META NAME="APPLICATION-NAME" CONTENT="Long headed application">
</HEAD>
```
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 email when HTML is used as an email authoring format, the `title` element can be omitted.

### 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

#### Accessibility considerations:
- For authors
- For implementers

#### DOM interface:

```idl
[Export=Window]
interface HTMLOTitleElement : HTMLElement {
    [HTMLConstructor] constructor();
    [[CEReactions]] 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.

For web developers (non-normative)

**title**. **text** [ = value ]

Returns the child text content of the element.

Can be set, to replace the element's children with the given value.

The `text` attribute's getter must return this `title` element's child text content.

The `text` attribute's setter must `string replace all` with the given value within this `title` element.
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>
...
<h1>The Dances</h1>
```

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 `form submission`

**Accessibility considerations**: For authors, For implementers.

**DOM interface**:

```
IDL [Exposed=Window]
interface HTMLBaseElement : HTMLElement {
  [HTMLConstructor] constructor();

  [CEReactions] attribute USVString href;
  [CEReactions] attribute DOMString target;
};
```

The `base` element allows authors to specify the `document base URL` for the purposes of `parsing URL`, 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 (its `manifest` attribute isn't affected by `base` elements).

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.

To get an element's target, given an `a`, `area`, or `form` element `element`, run these steps:

1. If `element` has a `target` attribute, then return that attribute's value.
2. If `element`'s node document contains a `base` element with a `target` attribute, then return the value of the `target` attribute of the first such `base` element.
3. Return the empty string.

A `base` element that is the first `base` element with an `href` content attribute in a document tree 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` attribute with `document`'s fallback base URL, and `document`'s character encoding. (Thus, the `base` element isn't affected by itself.)
3. Set `element`'s 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 this 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

In this example, a `base` element is used to set the document base URL:
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
  - If the element is allowed in the body, flow content.
  - If the element is allowed in the body, phrasing 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.
- If the element is allowed in the body, where phrasing content is expected.

**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 between the document containing the hyperlink and the destination resource.
  - `media` — Applicable media.
  - `integrity` — Integrity metadata used in Subresource Integrity checks (SRI).
  - `hreflang` — Language of the linked resource.
  - `type` — Hint for the type of the referenced resource.
  - `referrerpolicy` — Referrer policy for fetches initiated by the element.
  - `sizes` — Sizes of the icons (for rel="icon").
  - `imagesrcset` — Images to use in different situations, e.g., high-resolution displays, small monitors, etc. (for rel="preload").
  - `imagesizes` — Image sizes for different page layouts (for rel="preload").
  - `as` — Potential destination for a preload request (for rel="preload" and rel="modulepreload").
  - `color` — Color to use when customizing a site's icon (for rel="mask-icon").
  - `disabled` — Whether the link is disabled.
- Also, the `title` attribute has special semantics on this element: Title of the link; CSS style sheet set name.

**Accessibility considerations:**
- For authors.
- For implementers.

**DOM interface:**

```html
IDL [Exposed=Window]
interface HTMLLinkElement : HTMLElement {
    [HTMLConstructor] constructor();
}
```
The link element allows authors to link their document to other resources.

The address of the link(s) is given by the href attribute. If the href attribute is present, then its value must be a valid non-empty URL potentially surrounded by spaces. One or both of the href or imagesrcset attributes must be present.

If both the href and imagesrcset attributes are absent, then the element does not define a link.

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 an unordered set of unique 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.

A link element must have either a rel attribute or an itemprop attribute, but not both.

If a link element has an itemprop attribute, or has a rel attribute that contains only keywords that are body-ok, then the element is said to be allowed in the body. This means that the element can be used where phrasing content expected.

If the rel attribute is used, the element can only sometimes be used in the body of the page. When used with the itemprop attribute, the element can be used both in the head element and in the body of the page, subject to the constraints of the microdata model.

Two categories of links can be created using the link element: links to external resources and hyperlinks. The 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.
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.

For example, the following `<link>` element creates two hyperlinks (to the same page):

```html
<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.

Hyperlinks created with the `<link>` element and its rel attribute apply to the whole document. 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.

Unlike those created by a and area elements, hyperlinks created by `<link>` elements are not displayed as part of the document by default, in user agents that support the suggested default rendering. And even if they are force-displayed using CSS, they have no activation behavior. Instead, they primarily provide semantic information which might be used by the page or by other software that consumes the page's contents. Additionally, the user agent can provide its own UI for following such hyperlinks.

The exact behavior for links to external resources depends on the exact relationship, as defined for the relevant link type.

The crossorigin attribute is a CORS settings attribute. It is intended for use with external resource links.

The media attribute says which media the resource applies to. The value must be a valid media query list.

The integrity attribute represents the integrity metadata for requests which this element is responsible for. The value is text. The attribute must only be specified on `<link>` elements that have a rel attribute that contains the stylesheet, preload, or modulepreload keyword. [SRI]

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 string.

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.

The referrer policy attribute is a referrer policy attribute. It is intended for use with external resource links, where it helps set the referrer policy used when fetching and processing the linked resource. [REFERRERPOLICY]

The title attribute gives the title of the link. With one exception, it is purely advisory. The value is text. The exception is for stylesheet links that are in a document tree, for which the title attribute defines CSS style sheet sets.

The imagesrcset attribute may be present, and is a srcset attribute.

The imagesrcset and href attributes (if width descriptors are not used) together contribute the imagesizes attribute to the source set.

If the imagesrcset attribute is present and has any image candidate strings using a width descriptor, the imagesizes attribute must also be present, and is a sizes attribute. The imagesizes attribute contributes the source size to the source.
The `imagesrcset` and `imagesizes` attributes must only be specified on `link` elements that have both a `rel` attribute that specifies the `preload` keyword, as well as an `as` attribute in the "image" state.

Example

These attributes allow preloading the appropriate resource that is later used by an `img` element that has the corresponding values for its `srcset` and `sizes` attributes:

```html
<link rel="preload" as="image"
  imagesrcset="wolf_400px.jpg 400w, wolf_800px.jpg 800w, wolf_1600px.jpg 1600w"
  imagesizes="50vw">
<!-- ... later, or perhaps inserted dynamically ... -->
<img src="wolf.jpg" alt="A rad wolf"
  srcset="wolf_400px.jpg 400w, wolf_800px.jpg 800w, wolf_1600px.jpg 1600w"
  sizes="50vw">
```

Note how we omit the `href` attribute, as it would only be relevant for browsers that do not support `imagesrcset`, and in those cases it would likely cause the incorrect image to be preloaded.

Example

The `imagesrcset` attribute can be combined with the `media` attribute to preload the appropriate resource selected from a `picture` element's sources, for `art direction`:

```html
<link rel="preload" as="image"
  imagesrcset="dog-cropped-1x.jpg, dog-cropped-2x.jpg 2x"
  media="(max-width: 800px)">
<link rel="preload" as="image"
  imagesrcset="dog-wide-1x.jpg, dog-wide-2x.jpg 2x"
  media="(min-width: 801px)">
<!-- ... later, or perhaps inserted dynamically ... -->
<picture>
  <source srcset="dog-cropped-1x.jpg, dog-cropped-2x.jpg 2x"
    media="(max-width: 800px)">
  <img src="dog-wide-1x.jpg" srcset="dog-wide-2x.jpg 2x"
    alt="An awesome dog">
</picture>
```

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 attribute must only be specified on `link` elements that have a `rel` attribute that specifies the `icon` keyword or the `apple-touch-icon` keyword.

Note

The `apple-touch-icon` keyword is a registered extension to the predefined set of link types, but user agents are not required to support it in any way.

The `as` attribute specifies the potential destination for a preload request for the resource given by the `href` attribute. It is an enumerated attribute. Each potential destination is a keyword for this attribute, mapping to a state of the same name. The attribute must be specified on `link` elements that have a `rel` attribute that contains the `preload` keyword. It may be specified on `link` elements that have a `rel` attribute that contains the `modulepreload` keyword; in such cases it must have a value which is a script-like destination. For other `link` elements, it must not be specified.
The processing model for how the `as` attribute is used is given in an individual link type's fetch and process the linked resource algorithm.

**Note**
The attribute does not have a missing value default or invalid value default, meaning that invalid or missing values for the attribute map to no state. This is accounted for in the processing model. For preload links, both conditions are an error; for modulepreload links, a missing value will be treated as "script".

The `color` attribute is used with the `mask-icon` link type. The attribute must only be specified on `link` elements that have a `rel` attribute that contains the `mask-icon` keyword. The value must be a string that matches the CSS `<color>` production, defining a suggested color that user agents can use to customize the display of the icon that the user sees when they pin your site.

**Note**
This specification does not have any user agent requirements for the `color` attribute.

**Note**
The `mask-icon` keyword is a registered extension to the predefined set of link types, but user agents are not required to support it in any way.

`link` elements have an associated explicitly enabled boolean. It is initially false.

The `disabled` attribute is a boolean attribute that is used with the `stylesheet` link type. The attribute must only be specified on `link` elements that have a `rel` attribute that contains the `stylesheet` keyword.

Whenever the `disabled` attribute is removed, set the `link` element's explicitly enabled attribute to true.

**Example**
Removing the `disabled` attribute dynamically, e.g., using `document.querySelector("link").removeAttribute("disabled")`, will fetch and apply the style sheet:

```html
<link disabled rel="alternate stylesheet" href="css/pooh"/>
```

The IDL attributes `href`, `hreflang`, `integrity`, `media`, `rel`, `sizes`, `type`, and `disabled` each must reflect the respective content attributes of the same name.

**Note**
There is no reflecting IDL attribute for the `color` attribute, but this might be added later.

The `as` IDL attribute must reflect the `as` content attribute, limited to only known values.

The `crossOrigin` IDL attribute must reflect the `crossorigin` content attribute, limited to only known values.

The `referrerPolicy` IDL attribute must reflect the `referrerPolicy` content attribute, limited to only known values.

The `imageSrcset` IDL attribute must reflect the `imagesrcset` content attribute.

The `imageSizes` IDL attribute must reflect the `imagesizes` content attribute.

The `relList` IDL attribute must reflect the `rel` content attribute.

4.2.4.1 Processing the `media` attribute

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.

The default, if the media attribute is omitted, is "all", meaning that by default links apply to all media.

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.

4.2.4.2 Processing the type attribute

If the type 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 string, 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 UA does not support the given MIME type for the given link relationship, then the UA should not fetch and process the linked resource; if the UA does support the given MIME type for the given link relationship, then the UA should fetch and process the linked 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 fetch and process the linked resource if the type was known and supported, then the user agent should fetch and process the linked 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

If a document contains style sheet links labeled as follows:

```html
<link rel="stylesheet" href="A" type="text/plain"/>
<link rel="stylesheet" href="B" type="text/css"/>
<link rel="stylesheet" href="C"/>
```

...then a compliant UA 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.
4.2.4.3 Fetching and processing a resource from a link element

All external resource links\(^{284}\) have a fetch and process the linked resource algorithm, which takes a link element \(el\). They also have linked resource fetch setup steps which take a link element \(el\) and request \(request\). Individual link types may provide their own fetch and process the linked resource algorithm, but unless explicitly stated, they use the default fetch and process the linked resource\(^{164}\) algorithm. Similarly, individual link types may provide their own linked resource fetch setup steps, but unless explicitly stated, these steps just return true.

The default fetch and process the linked resource, given a link element \(el\), is as follows:

1. If \(el\)'s href attribute's value is the empty string, then return.
2. Parse a URL given \(el\)'s href attribute, relative to \(el\)'s node document. If that fails, then return. Otherwise, let url be the resulting URL record\(^{89}\).
3. Let corsAttributeState be the current state of \(el\)'s crossorigin content attribute.
4. Let request be the result of creating a potential-CORS request given url, the empty string, and corsAttributeState.
5. Set request's synchronous flag.
6. Set request's client to \(el\)'s node document's relevant settings object\(^{910}\).
7. Set request's cryptographic nonce metadata to the current value of \(el\)'s [[CryptographicNonce]]\(^{92}\) internal slot.
8. Set request's integrity metadata to the current value of \(el\)'s integrity content attribute.
9. Set request's referrer policy to the current state of \(el\)'s referrerpolicy attribute.
10. Run the linked resource fetch setup steps\(^{164}\), given \(el\) and request. If the result is false, then return.
11. Run the following steps in parallel\(^{942}\):
   1. Let response be the result of fetching request.
   2. Let success be true.
   3. If response is a network error or its status is not an ok status, set success to false.

   **Note**

   Note that content-specific errors, e.g., CSS parse errors or PNG decoding errors, do not affect success.

4. If success is true, wait for the link resource's critical subresources\(^{44}\) to finish loading.

The specification that defines a link type's critical subresources\(^{44}\) (e.g., CSS) is expected to describe how these subresources are fetched and processed. However, since this is not currently explicit, this specification describes waiting for a link resource's critical subresources\(^{44}\) to be fetched and processed, with the expectation that this will be done correctly.

5. Queue an element task\(^{932}\) on the networking task source\(^{939}\) given \(el\) to process the linked resource\(^{164}\) given \(el\), success, and response.

User agents may opt to only try to fetch and process\(^{164}\) such resources when they are needed, instead of pro-actively fetching all the external resources\(^{284}\) that are not applied.

Similar to the fetch and process the linked resource\(^{164}\) algorithm, all external resource links\(^{284}\) have a process the linked resource algorithm which takes a link element \(el\), boolean success, and response \(response\). Individual link types may provide their own process the linked resource\(^{164}\) algorithm, but unless explicitly stated, that algorithm does nothing.

Unless otherwise specified for a given rel keyword, the element must delay the load event\(^{167}\) of the element's node document until all the attempts to fetch and process the linked resource\(^{164}\) and its critical subresources\(^{44}\) are complete. (Resources that the user agent has not yet attempted to fetch and process, e.g., because it is waiting for the resource to be needed, do not delay the load event\(^{167}\).)
4.2.4.4 Processing `Link` headers

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][166] [WEblink][167]

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 processing of `Link` headers, in particular their influence on a Document's script-blocking style sheet counter, is not defined. See issue #4224 for discussion on integrating this into the spec.

4.2.4.5 Providing users with a means to follow hyperlinks created using the `link` element

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).

4.2.5 The `meta` element

Categories:
- Metadata content
- If the `itemprop` attribute is present: flow content
- If the `itemprop` attribute is present: phrasing 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.
- If the `itemprop` attribute is present: where metadata content is expected.
- If the `itemprop` attribute is present: where phrasing 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
Accessibility considerations:

- For authors.
- For implementers.

DOM interface:

```javascript
[Exposed=Window]
interface HTMLMetaElement : HTMLElement {
    [HTMLConstructor] constructor();

    [CEReactions] attribute DOMString name;
    [CEReactions] attribute DOMString httpEquiv;
    [CEReactions] attribute DOMString content;

    // also has obsolete members
};
```

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`, `charset`, and `itemprop` attributes must be specified.

If either `name`, `http-equiv`, or `itemprop` 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, its value must be an ASCII case-insensitive match for the string "utf-8".

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 where the `name` attribute value is an ASCII case-insensitive match for `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 `document 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` where the `name` attribute value is an ASCII case-insensitive match for `application-name` and whose `language` is the language in question.

If none of the languages have such a `meta` element, then return; 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 where the `name` attribute value is an ASCII case-insensitive match for `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 where the `name` attribute value is an ASCII case-insensitive match for `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**

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:

```html
<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**

This page about typefaces on British motorways uses a `meta` element to specify some keywords that users might use to look for the page:

```html
<!DOCTYPE HTML>
<html lang="en-GB">
<head>
```

167
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 \( \text{keywords} \) be an empty list.

2. For each \( \text{meta} \) element with a \( \text{name} \) attribute and a \( \text{content} \) attribute and where the \( \text{name} \) attribute value is an ASCII case-insensitive match for \( \text{keywords} \):
   
   1. Split the value of the element's \( \text{content} \) attribute on commas.
   
   2. Add the resulting tokens, if any, to \( \text{keywords} \).

3. Remove any duplicates from \( \text{keywords} \).

4. Return \( \text{keywords} \). 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.

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 \( \text{keywords} \) be an empty list.

2. For each \( \text{meta} \) element with a \( \text{name} \) attribute and a \( \text{content} \) attribute and where the \( \text{name} \) attribute value is an ASCII case-insensitive match for \( \text{keywords} \):
   
   1. Split the value of the element's \( \text{content} \) attribute on commas.
   
   2. Add the resulting tokens, if any, to \( \text{keywords} \).

3. Remove any duplicates from \( \text{keywords} \).

4. Return \( \text{keywords} \). 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

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.

referrer

The value must be a \texttt{referrer policy}, which defines the default \texttt{referrer policy} for the \texttt{Document}. If any \( \text{meta} \) elements are inserted into the document or removed from the document, or existing \( \text{meta} \) elements have their \( \text{name} \) or \( \text{content} \) attributes changed, user agents must run the following algorithm:

1. Let \( \text{candidate elements} \) be the list of all \( \text{meta} \) elements that meet the following criteria, in tree order:
   
   ◦ The element is in a document tree
   
   ◦ The element has a \( \text{name} \) attribute, whose value is an ASCII case-insensitive match for \( \text{referrer} \).

   ◦ The element has a \( \text{content} \) attribute, whose value is not the empty string

   ◦ The element is a child of the \texttt{head element} of the document

2. For each \texttt{element} in \texttt{candidate elements}:

   1. Let \texttt{value} be the value of \texttt{element}'s \texttt{content} attribute, converted to ASCII lowercase.

   2. If \texttt{value} is one of the values given in the first column of the following table, then set \texttt{value} to the value given in the second column:

<table>
<thead>
<tr>
<th>Legacy value</th>
<th>Referrer policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>never</td>
<td>no-referrer</td>
</tr>
<tr>
<td>default</td>
<td>the default referrer policy</td>
</tr>
<tr>
<td>always</td>
<td>unsafe-url</td>
</tr>
<tr>
<td>origin-when-crossorigin</td>
<td>origin-when-cross-origin</td>
</tr>
</tbody>
</table>

3. If \texttt{value} is a \texttt{referrer policy}, then set \texttt{element}'s \texttt{node document}'s \texttt{referrer policy} to \texttt{policy}.

Note

The fact that these steps are applied for each element enables deployment of fallback values for older user agents.
**theme-color**

The value must be a string that matches the CSS `<color>` production, defining a suggested color that user agents should use to customize the display of the page or of the surrounding user interface. For example, a browser might color the page’s title bar with the specified value, or use it as a color highlight in a tab bar or task switcher.

There must not be more than one `meta` element with its `name` attribute value set to an ASCII case-insensitive match for `theme-color` per document.

**Example**

This standard itself uses “WHATWG green” as its theme color:

```html
<!DOCTYPE HTML>
<title>HTML Standard</title>
<meta name="theme-color" content="#3c790a">
```

To obtain a page’s theme color, user agents must run the following steps:

1. Let `candidate elements` be the list of all `meta` elements that meet the following criteria, in tree order:
   - The element is in a document tree
   - The element has a `name` attribute, whose value is an ASCII case-insensitive match for `theme-color`
   - The element has a `content` attribute
2. For each `element` in `candidate elements`:
   1. Let `value` be the result of stripping leading and trailing ASCII whitespace from the value of `element`’s `content` attribute.
   2. Let `color` be the result of parsing `value`.
   3. If `color` is not failure, then return `color`.
3. Return nothing (the page has no theme color).

If any `meta` elements are inserted into the document or removed from the document, or existing `meta` elements have their `name` or `content` attributes changed, user agents must re-run the above algorithm and apply the result to any affected UI.

When using the theme color in UI, user agents may adjust it in implementation-specific ways to make it more suitable for the UI in question. For example, if a user agent intends to use the theme color as a background and display white text over it, it might use a darker variant of the theme color in that part of the UI, to ensure adequate contrast.

**color-scheme**

To aid user agents in rendering the page background with the desired color scheme immediately (rather than waiting for all CSS in the page to load), a ‘`color-scheme`’ value can be provided in a `meta` element.

The value must be a string that matches the syntax for the CSS ‘`color-scheme`’ property value, and will be cascaded with the color-scheme declarations for the document root element as described below.

There must not be more than one `meta` element with its `name` attribute value set to an ASCII case-insensitive match for `color-scheme` per document.

**Example**

The following declaration indicates that the page is aware of and can handle a color scheme with dark background colors and light foreground colors:

```html
<meta name="color-scheme" content="dark">
```

To obtain a page’s color-scheme, user agents must run the following steps:
1. Let candidate elements be the list of all \texttt{meta} elements that meet the following criteria, in tree order:
   
   ◦ The element is in a document tree
   ◦ The element has a \texttt{name} attribute, whose value is an ASCII case-insensitive match for \texttt{color-scheme}
   ◦ The element has a \texttt{content} attribute

2. For each element in candidate elements:
   
   1. Let parsed be the result of parsing a list of component values given the value of element's \texttt{content} attribute.
   2. If parsed is a valid CSS 'color-scheme' property value, then treat that value as a declaration of the CSS 'color-scheme' property on element's root, cascaded as a presentational hint, and return.

If any \texttt{meta} elements are inserted into the document or removed from the document, or existing \texttt{meta} elements have their \texttt{name} or \texttt{content} attributes changed, user agents must re-run the above algorithm.

\textbf{Note}

Because these rules check successive elements until they find a match, an author can provide multiple such values to handle fallback for legacy user agents. Opposite to how CSS fallback works for properties, the multiple meta elements needs to be arranged with the legacy values after the newer values.

**4.2.5.2 Other metadata names**

Anyone can create and use their own extensions to the predefined set of metadata names. There is no requirement to register such extensions.

However, a new metadata name should not be created in any of the following cases:

- If either the name is a URL, or the value of its accompanying \texttt{content} attribute is a URL; in those cases, registering it as an extension to the predefined set of link types is encouraged (rather than creating a new metadata name).
- If the name is for something expected to have processing requirements in user agents; in that case it ought to be standardized.

Also, before creating and using a new metadata name, consulting the \texttt{WHATWG Wiki MetaExtensions page} is encouraged — to avoid choosing a metadata name that's already in use, and to avoid duplicating the purpose of any metadata names that are already in use, and to avoid new standardized names clashing with your chosen name. \texttt{[WHATWGWIKI]}

Anyone is free to edit the WHATWG Wiki MetaExtensions page at any time to add a metadata name. New metadata names can 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 added in the "proposed" state for a period of a month or more without being used or specified, then it may be removed from the WHATWG Wiki MetaExtensions page.

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.

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.

<table>
<thead>
<tr>
<th>State</th>
<th>Keyword</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content Language</td>
<td>content-language</td>
<td>Non-conforming</td>
</tr>
<tr>
<td>Encoding declaration</td>
<td>content-type</td>
<td></td>
</tr>
<tr>
<td>Default style</td>
<td>default-style</td>
<td></td>
</tr>
<tr>
<td>Refresh</td>
<td>refresh</td>
<td></td>
</tr>
<tr>
<td>Set-Cookie</td>
<td>set-cookie</td>
<td>Non-conforming</td>
</tr>
<tr>
<td>X-UA-Compatible</td>
<td>x-ua-compatible</td>
<td></td>
</tr>
<tr>
<td>Content security policy</td>
<td>content-security-policy</td>
<td></td>
</tr>
</tbody>
</table>

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")

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 return.
2. If the element's content attribute contains a U+002C COMMA character (,), then return.
3. Let input be the value of the element's content attribute.
4. Let position point at the first character of input.
5. Skip ASCII whitespace within input given position.
6. Collect a sequence of code points that are not ASCII whitespace from input given position.
7. Let candidate be the string that resulted from the previous step.
8. If candidate is the empty string, return.

9. Set the **pragma-set default language**\(^1\) to candidate.

Note

If the value consists of multiple space-separated tokens, tokens after the first are ignored.

**Encoding declaration state** (http-equiv="content-type")\(^1\)

The **Encoding declaration state**\(^2\) 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 **ASCII whitespace**, followed by the literal string "charset=utf-8".

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**, but elements with an **http-equiv** attribute in that state must not be used in **XML documents**.

**Default style state** (http-equiv="default-style")\(^1\)

This pragma sets the **name** of the default **CSS style sheet set**.

1. If the **meta** element has no **content** attribute, or if that attribute's value is the empty string, then return.

2. Change the preferred CSS style sheet set name with the name being the value of the element's **content** attribute. \([CSSOM]\)\(^1\)

**Refresh state** (http-equiv="refresh")\(^1\)

This pragma acts as timed redirect.

A **Document** object has an associated **will declaratively refresh** (a boolean). It is initially false.

1. If the **meta** element has no **content** attribute, or if that attribute's value is the empty string, then return.

2. Let **input** be the value of the element's **content** attribute.

3. Run the **shared declarative refresh steps** with the **meta** element's **node document**, **input**, and the **meta** element.

The **shared declarative refresh steps**, given a **Document** object **document**, string **input**, and optionally a **meta** element **meta**, are as follows:

1. If **document's will declaratively refresh** is true, then return.

2. Let **position** point at the first **code point** of **input**.

3. **Skip ASCII whitespace** within **input** given **position**.

4. Let **time** be 0.

5. Collect a sequence of code points that are **ASCII digits** from **input** given **position**, and let the result be **timeString**.

6. If **timeString** is the empty string, then:

   1. If the **code point** in **input** pointed to by **position** is not U+002E (.), then return.

7. Otherwise, set **time** to the result of parsing **timeString** using the **rules for parsing non-negative integers**\(^6\).

8. Collect a sequence of code points that are **ASCII digits** and U+002E FULL STOP characters (.) from **input** given **position**. Ignore any collected characters.

9. Let **urlRecord** be **document's URL**.
10. If `position` is not past the end of `input`, then:

1. If the code point in `input` pointed to by `position` is not U+003B (;), U+002C (,), or ASCII whitespace, then return.
2. Skip ASCII whitespace within `input` given `position`.
3. If the code point in `input` pointed to by `position` is U+003B (;) or U+002C (,), then advance `position` to the next code point.
4. Skip ASCII whitespace within `input` given `position`.

11. If `position` is not past the end of `input`, then:

1. Let `urlString` be the substring of `input` from the code point at `position` to the end of the string.
2. If the code point in `input` pointed to by `position` is U+0055 (U) or U+0075 (u), then advance `position` to the next code point. Otherwise, jump to the step labeled `skip quotes`.
3. If the code point in `input` pointed to by `position` is U+0052 (R) or U+0072 (r), then advance `position` to the next code point. Otherwise, jump to the step labeled `parse`.
4. If the code point in `input` pointed to by `position` is U+004C (L) or U+006C (l), then advance `position` to the next code point. Otherwise, jump to the step labeled `parse`.
5. Skip ASCII whitespace within `input` given `position`.
6. If the code point in `input` pointed to by `position` is U+003D (=), then advance `position` to the next code point.
7. Otherwise, jump to the step labeled `parse`.
8. Skip quotes: If the code point in `input` pointed to by `position` is U+0027 (') or U+0022 ("), then let `quote` be that code point, and advance `position` to the next code point. Otherwise, let `quote` be the empty string.
9. Set `urlString` to the substring of `input` from the code point at `position` to the end of the string.
10. If `quote` is not the empty string, and there is a code point in `urlString` equal to `quote`, then truncate `urlString` at that code point, so that it and all subsequent code points are removed.
11. Parse: Parse `urlString` relative to `document`. If that fails, return. Otherwise, set `urlRecord` to the resulting URL record.

12. Set `document`'s will declaratively refresh to true.

13. 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 `meta` is given, `document`'s active sandboxing flag set does not have the sandboxed automatic features browsing context flag set, then navigate document's browsing context to `urlRecord`, with historyHandling set to "replace" and the source browsing context set to document's 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 `document`'s completely loaded time, adjusted to take into account user or user agent preferences.
- If `meta` is given, at least `time` seconds have elapsed since `meta` was inserted into the document, `document`, adjusted to take into account user or user agent preferences.

Note: It is important to use `document` here, and not `meta`'s node document, as that might have changed between the initial set of steps and the refresh coming due and meta is not always given (in case of the HTTP `Refresh` header).

- Provide the user with an interface that, when selected, navigates a browsing context to `urlRecord`, with `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 ASCII whitespace, 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 string 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**

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:

```html
<meta http-equiv="Refresh" content="300">  
```

**Example**

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:

```html
<meta http-equiv="Refresh" content="20; URL=page4.html">  
```

**Set-Cookie state**

This pragma is non-conforming and has no effect.

User agents are required to ignore this pragma.

**X-UA-Compatible state**

In practice, this pragma encourages Internet Explorer to more closely follow the specifications.

For meta elements with an http-equiv attribute in the X-UA-Compatible state, the content attribute must have a value that is an ASCII case-insensitive match for the string "IE=edge".

User agents are required to ignore this pragma.

**Content security policy state**

This pragma enforces a Content Security Policy on a Document. [CSP]

1. If the meta element is not a child of a head element, return.
2. If the meta element has no content attribute, or if that attribute's value is the empty string, then return.
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, with a source of "meta", and a disposition of "enforce".
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. [CSP]

**Example**

A page might choose to mitigate the risk of cross-site scripting attacks by preventing the execution of inline JavaScript, as well...
There must not be more than one `meta` element with any particular state in the document at a time.

### 4.2.5.4 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 Encoding standard requires use of the UTF-8 character encoding and requires use of the "utf-8" encoding label to identify it. Those requirements necessitate that the document's character encoding declaration, if it exists, specifies an encoding label using an ASCII case-insensitive match for "utf-8". Regardless of whether a character encoding declaration is present or not, the actual character encoding used to encode the document must be UTF-8.

To enforce the above rules, authoring tools must default to using UTF-8 for newly-created documents.

The following restrictions also apply:

- 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 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 when all characters are in the ASCII range, 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.

Using non-UTF-8 encodings can have unexpected results on form submission and URL encodings, which use the document's character encoding by default.

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.)

In XML, the XML declaration should be used for inline character encoding information, if necessary.

**Example**

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):

```html
<meta charset="utf-8"/>
```

In XML, the XML declaration would be used instead, at the very top of the markup:

```xml
<?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:
- Text that gives a conformant style sheet.

Tag omission in text/html:
- Neither tag is omissible.

Content attributes:
- Global attributes:
  - media — Applicable media
  - Also, the title attribute has special semantics on this element: CSS style sheet name.

Accessibility considerations:
- For authors.
- For implementers.

DOM interface:

```idl
[Exposed=Window]
interface HTMLStyleElement : HTMLElement {
  [HTMLConstructor] constructor();

  [CEReactions] attribute DOMString media;
  // also has obsolete members
};
```

HTMLStyleElement includes LinkStyle;

The **style** element allows authors to embed CSS style sheets 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 **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.

The default, if the **media** attribute is omitted, is "all", meaning that by default styles apply to all media.

The **title** attribute on **style** elements defines CSS 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. If the **style** element is not in a document tree, then the **title** attribute is ignored. [CSSOM]

**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 user agent must run the update a style block algorithm 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 becomes connected or disconnected.

• The element's children changed steps run.

The update a style block algorithm 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 connected, then return.

4. If element's type attribute is present and its value is neither the empty string nor an ASCII case-insensitive match for "text/css", then return.

   Note
   In particular, a type value with parameters, such as "text/css; charset=utf-8", will cause this algorithm to return early.

5. 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 child text content, then return. [CSP]

6. 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. CSSOM defines what happens when the attribute is dynamically set, changed, or removed.

   title
   The title attribute of element, if element is in a document tree, or the empty string otherwise.

   Note
   Again, this is a reference to the attribute.

   alternate flag
   Unset.

   origin-clean flag
   Set.

   location
   parent CSS style sheet
   owner CSS rule
     null

   disabled flag
   Left at its default value.

   CSS rules
   Left uninitialized.
Once the attempts to obtain the style sheet's critical subresources\[^{44}\], if any, are complete, or, if the style sheet has no critical subresources\[^{44}\], once the style sheet has been parsed and processed, the user agent must run these steps:

1. Let \texttt{element} be the \texttt{style}\[^{176}\] element associated with the style sheet in question.
2. Let \texttt{success} be true.
3. If the attempts to obtain any of the style sheet's critical subresources\[^{44}\] failed for any reason (e.g., DNS error, HTTP 404 response, a connection being prematurely closed, unsupported Content-Type), set \texttt{success} to false.

4. Queue an element task\[^{932}\] on the networking task source\[^{939}\] given \texttt{element} and the following steps:
   1. If \texttt{success} is true, \texttt{fire an event} named load\[^{9126}\] at \texttt{element}.
   2. Otherwise, \texttt{fire an event} named error\[^{9126}\] at \texttt{element}.
   3. If \texttt{element} contributes a script-blocking style sheet\[^{179}\]:
      1. Assert: \texttt{element}'s node document's script-blocking style sheet counter\[^{179}\] is greater than 0.
      2. Decrement \texttt{element}'s node document's script-blocking style sheet counter\[^{179}\] by 1.

The element must delay the load event\[^{9157}\] of the element's node document until all the attempts to obtain the style sheet's critical subresources\[^{44}\], if any, are complete.

Note
\[^{95}\] Note that content-specific errors, e.g., CSS parse errors or PNG decoding errors, do not affect \texttt{success}.

Example
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.

```html
<!DOCTYPE html>
<html lang="en-US">
<head>
  <title>My favorite book</title>
  <style>
    body { color: black; background: white; }
    em { font-style: normal; color: red; }
  </style>
</head>
<body>
  <p>My favorite book of all time has got 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\[^{917}\]
If the style sheet referenced no other resources (e.g., it was an internal style sheet given by a \texttt{style}\[^{176}\] 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.

An element `el` in the context of a `Document` of an HTML parser or XML parser contributes a script-blocking style sheet if all of the following conditions are true:

- `el` was created by that `Document`'s parser.
- `el` is either a `style` element or a `link` element that was an external resource link that contributes to the styling processing model when the `el` was created by the parser.
- If the `el` is a `link` element, it's `media` attribute's value matches the environment.
- `el`'s style sheet was enabled when the element was created by the parser.
- The last time the event loop reached step 1, `el`'s root was that `Document`.
- The user agent hasn't given up on loading that particular style sheet yet. A user agent may give up on loading 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). Implementers 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.

It is expected that counterparts to the above rules also apply to `<?xml-stylesheet?>` PIs and HTTP `Link` headers. However, this has not yet been thoroughly investigated.

A `Document` has a script-blocking style sheet counter, which is a number, initially 0.

A `Document` has a style sheet that is blocking scripts if its script-blocking style sheet counter is greater than 0, or if that `Document` has a non-null browsing context whose container document is non-null and has a script-blocking style sheet counter greater than 0.

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 can be omitted if the element is empty, or if the first thing inside the `body` element is not ASCII whitespace 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 can be omitted if the `body` element is not immediately followed by a comment.
Content attributes:

- Global attributes:
  - `onafterprint`
  - `onbeforeprint`
  - `onbeforeunload`
  - `onhashchange`
  - `onlanguagechange`
  - `onmessage`
  - `onmessageerror`
  - `onoffline`
  - `ononline`
  - `onpagehide`
  - `onpageshow`
  - `onpopstate`
  - `onrejectionhandled`
  - `onstorage`
  - `onunhandledrejection`
  - `onunload`

Accessibility considerations:

- For authors.
- For implementers.

DOM interface:

```idl
[Exposed=Window]
interface HTMLBodyElement : HTMLElement {
    [HTMLConstructor] constructor();

    // also has obsolete members
};
```

The `body` element represents the contents 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 event handlers of the `Window` object named by the Window-reflecting body element event handler set, exposed on the `body` element, replace the generic event handlers with the same names normally supported by `HTMLElement`.

**Example**

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 attribute 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**

This page updates an indicator to show whether or not the user is online:
The `article` element

Categories:
- Flow content
- Sectioning content
- Palpable content

Contexts in which this element can be used:
Where sectioning content is expected.

Content model:
- Flow content

Tag omission in text/html:
Neither tag is omissible.

Content attributes:
- Global attributes

Accessibility considerations:
- For authors
- For implementers

DOM interface:
Uses `HTMLElement`.

The `article` element represents a complete, or self-contained, composition in a document, page, application, or site and that is, in principle, independently distributable or reusable, e.g. in syndication. This could be a forum post, a magazine or newspaper article, a blog entry, a user-submitted comment, an interactive widget or gadget, or any other independent item of content.

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 that accepts user-submitted comments could represent the comments as `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.

**Note**
When used specifically with content to be redistributed in syndication, the `article` element is similar in purpose to the entry element in Atom. [ATOM]
When the main content of the page (i.e. excluding footers, headers, navigation blocks, and sidebars) is all one single self-contained composition, that content may be marked with an article element, but it is technically redundant in that case (since it's self-evident that the page is a single composition, as it is a single document).

Example

This example shows a blog post using the article element, with some schema.org annotations:

```html
<article itemscope itemtype="http://schema.org/BlogPosting">
  <header>
    <h1 itemprop="headline">The Very First Rule of Life</h1>
    <p><time itemprop="datePublished" datetime="2009-10-09">3 days ago</time></p>
    <link itemprop="url" href="/?comments=0"/>
  </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>
  <footer>
    <a itemprop="discussionUrl" href="/?comments=1">Show comments... </a>
  </footer>
</article>

Here is that same blog post, but showing some of the comments:

```html
<article itemscope itemtype="http://schema.org/BlogPosting">
  <header>
    <h1 itemprop="headline">The Very First Rule of Life</h1>
    <p><time itemprop="datePublished" datetime="2009-10-09">3 days ago</time></p>
    <link itemprop="url" href="/"/>
  </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>
  <footer>
    <p>Posted by: <span itemprop="creator" itemtype="http://schema.org/Person">George Washington</span></p>
    <p><time itemprop="commentTime" datetime="2009-10-10">15 minutes ago</time></p>
  </footer>
  <p>Yeah! Especially when talking about your lobbyist friends!</p>
</article>

<article itemscope itemtype="http://schema.org/UserComments" id="c1">
  <link itemprop="url" href="#c1"/>
  <footer>
    <p>Posted by: <span itemprop="creator" itemtype="http://schema.org/Person">George Hammond</span></p>
    <p><time itemprop="commentTime" datetime="2009-10-10">5 minutes ago</time></p>
  </footer>
  <p>Hey, you have the same first name as me.</p>
</article>

<section>
  <h1>Comments</h1>
  <article itemscope itemtype="http://schema.org/UserComments" id="c2">
    <link itemprop="url" href="#c2"/>
    <footer>
      <p>Posted by: <span itemprop="creator" itemtype="http://schema.org/Person">George Hammond</span></p>
      <p><time itemprop="commentTime" datetime="2009-10-10">5 minutes ago</time></p>
    </footer>
    <p>Hey, you have the same first name as me.</p>
  </article>
</section>
```
Notice the use of footer to give the information for each comment (such as who wrote it and when): the footer element can appear at the start of its section when appropriate, such as in this case. (Using header in this case wouldn't be wrong either; it's mostly a matter of authoring preference.)

**Example**

In this example, article elements are used to host widgets on a portal page. The widgets are implemented as customized built-in elements in order to get specific styling and scripted behavior.

```html
<!DOCTYPE HTML>
<html lang=en>
<title>eHome Portal</title>
<script src="/scripts/widgets.js"></script>
<link rel=stylesheet href="/styles/main.css">
<article is="stock-widget">
  <h1>Stocks</h1>
  <table>
    <thead>
      <tr>
        <th>Stock</th>
        <th>Value</th>
        <th>Delta</th>
      </tr>
    </thead>
    <tbody template>
      <tr>
        <td><img></td>
        <strong></strong>
        <td></td>
        <td></td>
      </tr>
    </tbody>
  </table>

  <p><input type=button value="Refresh" onclick="this.parentElement.refresh()">
</article>
<article is="news-widget">
  <h1>News</h1>
  <ul template>
    <li><p><img><strong></strong></p></li>
    <p></p>
  </ul>

  <p><input type=button value="Refresh" onclick="this.parentElement.refresh()">
</article>
```

4.3.3 The **section** element

- **Categories**: Flow content, Sectioning content, Palpable content
- **Contexts in which this element can be used**: Where sectioning content is expected.
- **Content model**: Flow content
- **Tag omission in text/html**: Neither tag is omissible.
- **Content attributes**: Global attributes
- **Accessibility considerations**: For authors, For implementers.
- **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, typically with a heading.

**Example**

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 it would make sense to syndicate the contents of the element.

**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.

**Example**

In the following example, we see an article (part of a larger web page) about apples, containing two short sections.

```html
<article>
  <hgroup>
    <h1>Apples</h1>
    <h2>Tasty, delicious fruit!</h2>
  </hgroup>
  <p>The apple is the pomaceous fruit of the apple tree.</p>
  <section>
    <h1>Red Delicious</h1>
    <p>These bright red apples are the most common found in many supermarkets.</p>
  </section>
  <section>
    <h1>Granny Smith</h1>
    <p>These juicy, green apples make a great filling for apple pies.</p>
  </section>
</article>
```

Notice how the use of `section` means that the author can use `h1` elements throughout, without having to worry about whether a particular section is at the top level, the second level, the third level, and so on.

**Example**

Here is a graduation programme 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.)

```html
<!DOCTYPE Html>
<html lang="en">
  <head>
    <title>Graduation Ceremony Summer 2022</title>
  </head>
  <body>
    <h1>Graduation</h1>
    <section>
      <h1>Ceremony</h1>
    </section>
  </body>
</html>
```
Opening Procession

Speech by Validactorian

Speech by Class President

Presentation of Diplomas

Closing Speech by Headmaster

Graduates

- Molly Carpenter
- Anastasia Luccio
- Ebenezer McCoy
- Karrin Murphy
- Thomas Raith
- Susan Rodriguez

Example

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.
4.3.4 The nav element

Categories:
- Flow content
- Sectioning content
- Palpable content

Contexts in which this element can be used:
Where sectioning content is expected.

Content model:
- Flow content

Tag omission in text/html:
Neither tag is omissible.

Content attributes:
- Global attributes

Accessibility considerations:
For authors.
For implementers.

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.

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
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.
In the following example, the page has several places where links are present, but only one of those places is considered a navigation section.

Example
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

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:

```html
<nav>
  <h1>Navigation</h1>
  <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 thus 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

In this example, `nav` is used in an email application, to let the user switch folders:

```html
<p><input type=button value="Compose" onclick="compose()"></p>
<nav>
  <h1>Folders</h1>
```
4.3.5 The aside element

Categories:
- Flow content
- Sectioning content
- Palpable content

Contexts in which this element can be used:
Where sectioning content is expected.

Content model:
- Flow content

Tag omission in text/html:
Neither tag is omissible.

Content attributes:
- Global attributes

Accessibility considerations:
- For authors
- For implementers

DOM interface:
- Uses HTMLElement

The aside element represents a section of a page that consists of content that is tangentially related to the content around the aside element, 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 page.

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
The following example shows how an aside is used to mark up background material on Switzerland in a much longer news story on Europe.

```html
<aside>
  <h1>Switzerland</h1>
  <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>
```
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.</p>

<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>

<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

The following extract shows how <code>aside</code> can be used for blogrolls and other side content on a blog:

```html
<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>
      <h1>My blogroll</h1>
      <ul>
        <li><a href="https://blog.example.com/">Example Blog</a></li>
      </ul>
    </nav>
    <nav>
      <h1>Archives</h1>
      <ol reversed>
        <li><a href="/last-post">My last post</a></li>
        <li><a href="/first-post">My first post</a></li>
      </ol>
    </nav>
  </aside>
  <aside>
    <!-- this aside is tangentially related to the page also, it contains twitter messages from the blog author -->
    <h1>Twitter Feed</h1>
    <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 -->
  </article>
</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**: As a child of an **hgroup** element.

Where heading content is expected.

**Content model**: Phrasing content.

**Tag omission in text/html**: Neither tag is omissible.

**Content attributes**: Global attributes.

**Accessibility considerations**: For authors, For implementers.

**DOM interface**:

```idl
[Exposed=Window]
interface HTMLHeadingElement : HTMLElement {
    [HTMLConstructor] constructor();
```
These elements represent headings for their sections.

The semantics and meaning of these elements are defined in the section on headings and sections.

These elements have a rank given by the number in their name. The h1 element is said to have the highest rank, the h6 element has the lowest rank, and two elements with the same name have equal rank.

Example

As far as their respective document outlines (their heading and section structures) are concerned, these two snippets are semantically equivalent:

```html
<body>
<h1>Let's call it a drawing 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 drawing surface</h1>
<section>
<h1>Diving in</h1>
</section>
<section>
<h1>Simple shapes</h1>
</section>
<section>
<h1>Canvas coordinates</h1>
</section>
<section>
<h1>Canvas coordinates diagram</h1>
</section>
<h2>Paths</h2>
</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. This third snippet again has the same outline as the previous two:

```html
<body>
<h1>Let's call it a drawing surface</h1>
<section>
<h1>Diving in</h1>
</section>
<section>
<h1>Simple shapes</h1>
</section>
<section>
<h2>Canvas coordinates</h2>
</section>
</body>
```
4.3.7 The `<hgroup>` element

**Categories**
- Flow content
- Heading content
- Palpable content

**Contexts in which this element can be used**:
Where heading content is expected.

**Content model**:
One or more `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>` elements, optionally intermixed with script-supporting elements.

**Tag omission in text/html**:
Neither tag is omissible.

**Content attributes**
- Global attributes

**Accessibility considerations**:
- For authors
- For implementers

**DOM interface**:
Uses `HTMLElement`.

The `<hgroup>` element represents the heading of a section, which consists of all the `<h1>`–`<h6>` element children of the `<hgroup>` element. The element is used to group a set of `<h1>–<h6>` elements when the heading has multiple levels, such as subheadings, alternative titles, or taglines.

The rank of an `<hgroup>` element is the rank of the highest-ranked `<h1>`–`<h6>` element descendant of the `<hgroup>` element, if there are any such elements, or otherwise the same as for an `<h1>` element (the highest rank). Other `<h1>`–`<h6>` elements of heading content in the `<hgroup>` element indicate subheadings or subtitles or (secondary) alternative titles.

The section on headings and sections defines how `<hgroup>` elements are assigned to individual sections.

**Example**
Here are some examples of valid headings.

```html
<hgroup>
  <h1>The reality dysfunction</h1>
  <h2>Space is not the only void</h2>
</hgroup>

<hgroup>
  <h1>Dr. Strangelove</h1>
  <h2>Or: How I Learned to Stop Worrying and Love the Bomb</h2>
</hgroup>
```

The point of using `<hgroup>` in these examples is to prevent the `<h2>` element (which acts as a secondary title) from creating a
separate section of its own in any outline and to instead cause the contents of the h2 to be shown in rendered output from the outline algorithm in some way to indicate that it is not the title of a separate section but instead just a secondary title in a group of titles.

How a user agent exposes such multi-level headings in user interfaces (e.g. in tables of contents or search results) is left open to implementers, as it is a user interface issue. The first example above could be rendered as:

The reality dysfunction: Space is not the only void

Alternatively, it could look like this:

The reality dysfunction (Space is not the only void)

In interfaces where a title can be rendered on multiple lines, it could be rendered as follows, maybe with the first line in a bigger font size:

The reality dysfunction
Space is not the only void

Example

The following two examples show ways in which two h1 headings could be used within an hgroup element to group the US and UK names for the same movie.

```html
<hgroup>
  <h1>The Avengers</h1>
  <h1>Avengers Assemble</h1>
</hgroup>

<hgroup>
  <h1>Avengers Assemble</h1>
  <h1>The Avengers</h1>
</hgroup>
```

The first example above shows how the movie names might be grouped in a publication in the US, with the US name The Avengers as the (primary) title, and the UK name Avengers Assemble as the (secondary) alternative title. The second example above shows how the movie names might be grouped in a publication in the UK, with the UK name as the (primary) title, and the US name as the (secondary) alternative title.

In both cases it is important to note the use of the hgroup element to group the two titles indicates that the titles are not equivalent; instead the first h1 gives the (primary) title while the second gives the (secondary) alternative title. Even though both the title and alternative title are marked up with h1 elements, in a rendered view of output from the outline algorithm, the second h1 in the hgroup will be shown in some way that clearly indicates it is secondary; for example:

In a US publication:

The Avengers (Avengers Assemble)

In a UK publication:

Avengers Assemble (The Avengers)

Example

In the following example, an hgroup element is used to mark up a two-level heading in a wizard-style dialog box:

```html
<dialog onclose="walletSetup.continue(this.returnValue)">
  <hgroup>
    <h1>Wallet Setup</h1>
    <h2>Configure your Wallet funding source</h2>
  </hgroup>
</dialog>
```
A header element is intended to usually contain the section's heading (an h1-h6 element or an hgroup 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.
Here are some sample headers. This first one is for a game:

```html
<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:

```html
<header>
  <h1>Fullscreen API</h1>
  <h2>Living Standard — Last Updated 19 October 2015</h2>
  <dl>
    <dt>Participate:</dt>
    <dd><a href="https://github.com/whatwg/fullscreen">GitHub whatwg/fullscreen</a></dd>
    <dt>Commits:</dt>
    <dd><a href="https://github.com/whatwg/fullscreen/commits">GitHub whatwg/fullscreen/commits</a></dd>
  </dl>
</header>
```

Note
---
The `header` element is not sectioning content; it doesn't introduce a new section.

Example
---
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.

```html
<body>
  <header>
    <h1>Little Green Guys With Guns</h1>
    <nav>
      <ul>
        <li><a href="/games">Games</a></li>
        <li><a href="/forum">Forum</a></li>
        <li><a href="/download">Download</a></li>
      </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 -->
    <!-- this is still part of the subsection entitled "Games" -->
  </header>
  <p>You have three active games:</p>
  ...  
</body>
```

4.3.9 The `footer` element
---

Categories:
- Flow content
- Palpable content
Contexts in which this element can be used:\footnote{129}:
Where \texttt{flow content} is expected.

**Content model**:\footnote{129} \texttt{Flow content}, but with no \texttt{header} or \texttt{footer} element descendants.

**Tag omission in text/html**: Neither tag is omissible.

**Content attributes**:\footnote{130}:
Global attributes

**Accessibility considerations**:\footnote{130}:
If the nearest ancestor \texttt{sectioning content} or \texttt{sectioning root} element is \texttt{the body element}:
\begin{itemize}
  \item for authors;
  \item for implementers.
\end{itemize}
Otherwise:
\begin{itemize}
  \item for authors;
  \item for implementers.
\end{itemize}

**DOM interface**:\footnote{130}:
Uses \texttt{HTMLElement}.

The \texttt{footer} element \texttt{represents} a footer for its nearest ancestor \texttt{sectioning content} or \texttt{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 \texttt{footer} element contains entire sections, they \texttt{represent} appendices, indexes, long colophons, verbose license agreements, and other such content.

**Note**
Contact information for the author or editor of a section belongs in an \texttt{address} element, possibly itself inside a \texttt{footer}. Bylines and other information that could be suitable for both a \texttt{header} or a \texttt{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 \texttt{sectioning content} or \texttt{sectioning root} element is \texttt{the body element}, then it applies to the whole page.

**Note**
The \texttt{footer} element is not \texttt{sectioning content}; it doesn't introduce a new section.

**Example**
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>
  <hgroup>
    <h1>Lorem ipsum</h1>
    <h2>The ipsum of all lorems</h2>
  </hgroup>
  <p>A dolor sit amet, consectetur adipiscing 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>
```
Here is an example which shows the `footer` element being used both for a site-wide footer and for a section footer.

```html
<!DOCTYPE HTML>
<html lang="en">
  <head>
    <title>The Ramblings of a Scientist</title>
  </head>
  <body>
    <h1>The Ramblings of a Scientist</h1>
    <article>
      <h1>Episode 15</h1>
      <video src="/fm/015.ogv" controls preload>
        <a href="/fm/015.ogv">Download video</a>.</video>
      <footer>
        <p>Published <time datetime="2009-10-21T18:26-07:00">on 2009/10/21 at 6:26pm</time></p>
      </footer>
    </article>
    <article>
      <h1>My Favorite Trains</h1>
      <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>
        <p>Published <time datetime="2009-09-15T14:54-07:00">on 2009/09/15 at 2:54pm</time></p>
      </footer>
    </article>
    <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>
        <p>Copyright © 2009 Gordon Freeman</p>
      </nav>
    </footer>
  </body>
</html>
```

Example

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”:

```html
...<footer>
  <nav>
    <section>
      <h1>Articles</h1>
      <p><img src="images/somersaults.jpeg" alt=""> 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><img src="images/kindplus.jpeg" Tired of walking on the edge of a cliff? Our guest writer Lara shows you how to bumble your way through the bars. <a href="articles/kindplus/1">Read more...</a></p>
      <p><img src="images/crisps.jpeg" 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>
  </nav>
</footer>
```
4.3.10 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

**Accessibility considerations**:
- For authors
- For implementers

**DOM interface**:
Uses HTMLElement.

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**

For example, a page at the W3C web site related to HTML might include the following contact information:

```html
<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**

For example, the following is non-conforming use of the **address** element:

```html
<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**

In this example the footer contains contact information and a copyright notice.

```html
<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>
```

### 4.3.11 Headings and sections

The `h1`–`h6` elements and the `hgroup` element 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.

Certain elements are said to be `sectioning roots`, including `blockquote`, `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`, `dialog`, `fieldset`, `figure`, `td` 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**

For the following fragment:

```html
<body>
  <h1>Foo</h1>
  <h2>Bar</h2>
```
Sections may contain headings of any rank, but authors are strongly encouraged to either use only `h1` elements, or to use elements 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**

For example, the following is correct:

```html
<body>
  <h4>Apples</h4>
  <p>Apples are fruit.</p>
  <section>
    <h2>Taste</h2>
    <p>They taste lovely.</p>
    <h6>Sweet</h6>
    <p>Red apples are sweeter than green ones.</p>
    <h1>Color</h1>
    <p>Apples come in various colors.</p>
  </section>
</body>
```

However, the same document would be more clearly expressed as:

```html
<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>
  <section>
    <h2>Color</h2>
    <p>Apples come in various colors.</p>
  </section>
</body>
```
Both of the documents above are semantically identical and would produce the same outline in compliant user agents.

This third example is also semantically identical, and might be easier to maintain (e.g. if sections are often moved around in editing):

```html
<body>
  <h1>Apples</h1>
  <p>Apples are fruit.</p>
  <section>
    <h1>Taste</h1>
    <p>They taste lovely.</p>
  </section>
  <section>
    <h1>Sweet</h1>
    <p>Red apples are sweeter than green ones.</p>
  </section>
  <section>
    <h1>Color</h1>
    <p>Apples come in various colors.</p>
  </section>
</body>
```

This final example would need explicit style rules to be rendered well in legacy browsers. Legacy browsers without CSS support would render all the headings as top-level headings.

### 4.3.11.1 Creating an outline

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**

The following markup fragment:

```html
<body>
  <hgroup id="document-title">
    <h1>HTML</h1>
    <h2>Living Standard — Last Updated 12 August 2016</h2>
  </hgroup>
  <p>Some intro to the document.</p>
  <h2>Table of contents</h2>
  <ol id="toc">
  </ol>
  <h2>First section</h2>
  <p>Some intro to the first section.</p>
</body>
```

...results in the following outline being created for the body node (and thus the entire document):
1. **Section created for body element.**
   Associated with heading `<hgroup id="document-title">...</hgroup>` consisting of primary heading `<h1>HTML</h1>` and secondary heading `<h2>Living Standard — Last Updated 12 August 2016</h2>.
   Also associated with the paragraph `<p>Some intro to the document.</p>` (though it likely would not be shown in a rendered view of the outline).
   Nested sections:
   1. **Section implied for first h2 element.**
      Associated with heading `<h2>Table of contents</h2>.
      Also associated with the ordered list `<ol id=toc>...</ol>` (though it likely would not be shown in a rendered view of the outline).
      No nested sections.
   2. **Section implied for second h2 element.**
      Associated with heading `<h2>First section</h2>.
      Also associated with the paragraph `<p>Some intro to the first section.</p>` (though it likely would not be shown in a rendered view of the outline).
      No nested sections.

The following image shows what a rendered view of the outline might look like.

```
HTML: Living Standard — Last Updated 12 August 2016

- Table of contents
- First section
```

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, then:
      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`.

Note

If the element being entered is an `hgroup` element, that `hgroup` as a whole is a multi-level heading for the current section, with the highest-ranked `h1–h6` descendant of the `hgroup` providing the primary heading for the current section, and with other `h1–h6` descendants of the `hgroup` providing secondary headings 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
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.)

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 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

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 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

Consider the following snippet:

```xml
<body>
  <nav>
    <p><a href="/">Home</a></p>
  </nav>
</body>
```
This section is non-normative.

Note

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. [JAVASCRIPT]

```
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;
      }
    }
  }
}
```

4.3.11.2 Sample outlines

This section is non-normative.

Example

The following document shows a straight-forward application of the outline algorithm. First, here is the document, which is a book with very short chapters and subsections:

```
<!DOCTYPE HTML>
<html lang=en>
<title>The Tax Book (all in one page)</title>
<h1>The Tax Book</h1>
<h2>Earning money</h2>
```
Earning money is good.

Getting a job

To earn money you typically need a job.

Spending money

Spending is what money is mainly used for.

Cheap things

Buying cheap things often not cost-effective.

Expensive things

The most expensive thing is often not the most cost-effective either.

Investing money

You can lend your money to other people.

Losing money

If you spend money or invest money, sooner or later you will lose money.

Poor judgement

Usually if you lose money it's because you made a mistake.

This book would form the following outline:

1. The Tax Book
   1. Earning money
   2. Spending money
      1. Cheap things
      2. Expensive things
   3. Investing money
   4. Losing money
      1. Poor judgement

Notice that the title element does not participate in the outline.

Example

Here is a similar document, but this time using section elements to get the same effect:

<!DOCTYPE HTML>
<html lang=en>
<title>The Tax Book (all in one page)</title>
<h1>The Tax Book</h1>
<section>
<h1>Earning money</h1>
<p>Earning money is good.</p>
</section>
<section>
<h1>Getting a job</h1>
<p>To earn money you typically need a job.</p>
</section>
<section>
<h1>Spending money</h1>
<p>Spending is what money is mainly used for.</p>
</section>
<section>
<h1>Cheap things</h1>
<p>Buying cheap things often not cost-effective.</p>
</section>
<section>
<h1>Expensive things</h1>
<p>The most expensive thing is often not the most cost-effective either.</p>
</section>
<section>
<h1>Investing money</h1>
<p>You can lend your money to other people.</p>
</section>
Losing money

If you spend money or invest money, sooner or later you will lose money.

Poor judgement

Usually if you lose money it’s because you made a mistake.

This book would form the same outline:

1. The Tax Book
   1. Earning money
      1. Getting a job
   2. Spending money
      1. Cheap things
      2. Expensive things
   3. Investing money
   4. Losing money
      1. Poor judgement

A document can contain multiple top-level headings:

Example

Mixing both the \texttt{h1}–\texttt{p} model and the \texttt{section}–\texttt{h1} model can lead to some unintuitive results.

Consider for example the following, which is just the previous example but with the contents of the (implied) \texttt{body} wrapped in a \texttt{section}:

Example

The resulting outline would be:
1. (untitled page)
   1. Apples
   2. Bananas
   3. Carambola

This result is described as **unintuitive** because it results in three subsections even though there's only one section element. Effectively, the section is split into three, just like the implied body element in the previous example.

(In this example, "(untitled page)" is the implied heading for the body element, since it has no explicit heading.)

---

**Example**

Headings never rise above other sections. Thus, in the following example, the first h1 does not actually describe the page header; it describes the header for the second half of the page:

```html
<!DOCTYPE HTML>
<html lang="en">
<title>Feathers on The Site of Encyclopedic Knowledge</title>
<section>
<h1>A plea from our caretakers</h1>
<p>Please, we beg of you, send help! We're stuck in the server room!</p>
</section>
<h1>Feathers</h1>
<p>Epidermal growths.</p>
</html>
```

The resulting outline would be:

1. (untitled page)
   1. A plea from our caretakers
   2. Feathers

---

**Example**

Thus, when an article element starts with a nav block and only later has its heading, the result is that the nav block is not part of the same section as the rest of the article in the outline. For instance, take this document:

```html
<!DOCTYPE HTML>
<html lang="en">
<title>We're adopting a child! — Ray's blog</title>
<h1>Ray's blog</h1>
<article>
<header>
<nav>
<a href="?t=-1d">Yesterday</a>
<a href="?t=-7d">Last week</a>
<a href="?t=-1m">Last month</a>
</nav>
<h1>We're adopting a child!</h1>
</header>
<p>As of today, Janine and I have signed the papers to become the proud parents of baby Diane! We've been looking forward to this day for weeks.</p>
</article>
</html>
```

The resulting outline would be:

1. Ray's blog
   1. Untitled article
      1. Untitled navigation section
   2. We're adopting a child!

Also worthy of note in this example is that the header element has no effect whatsoever on the document outline.
The `hgroup` element can be used for subheadings. For example:

```html
<!DOCTYPE HTML>
<html lang="en">
<title>Chronotype: CS Student</title>
<hgroup>
<h1>The morning</h1>
<h2>06:00 to 12:00</h2>
</hgroup>
<p>We sleep.</p>
<hgroup>
<h1>The afternoon</h1>
<h2>12:00 to 18:00</h2>
</hgroup>
<p>We study.</p>
<hgroup>
<h2>Additional Commentary</h2>
<h3>Because not all this is necessarily true</h3>
<h6>Ok it's almost certainly not true</h6>
</hgroup>
<p>Yeah we probably play, rather than study.</p>
<hgroup>
<h1>The evening</h1>
<h2>18:00 to 00:00</h2>
</hgroup>
<p>We play.</p>
<hgroup>
<h1>The night</h1>
<h2>00:00 to 06:00</h2>
</hgroup>
<p>We play some more.</p>
</html>
```

The resulting outline would be:

1. The morning 06:00 to 12:00
2. The afternoon 12:00 to 18:00
   1. Additional Commentary Because not all this is necessarily true Ok it's almost certainly not true
3. The evening 18:00 to 00:00
4. The night 00:00 to 06:00

Exactly how this is represented by user agents, as most interface issues, is left as a matter of implementation preference, but the key part is that the `hgroup`'s descendant `h1`–`h6` elements are what form the element's heading. Thus, the following would be equally valid:

1. The morning — 06:00 to 12:00
2. The afternoon — 12:00 to 18:00
   1. Additional Commentary Because not all this is necessarily true — Ok it's almost certainly not true
3. The evening — 18:00 to 00:00
4. The night — 00:00 to 06:00

But so would the following:

1. The morning
2. The afternoon
   1. Additional Commentary
3. The evening
4. The night

The following would also be valid, though maybe less practical in most contexts:

1. The morning
   06:00 to 12:00
2. The afternoon
   12:00 to 18:00
   1. Additional Commentary
      Because not all this is necessarily true
4.3.11.3 Exposing outlines to users

User agents are encouraged to expose page outlines to users to aid in navigation. This is especially true for non-visual media, e.g. screen readers.

However, to mitigate the difficulties that arise from authors misusing sectioning content, user agents are also encouraged to offer a mode that navigates the page using heading content alone.

Example

For instance, a user agent could map the arrow keys as follows:

- **Shift+←** Left
  - Go to previous section, including subsections of previous sections
- **Shift+→** Right
  - Go to next section, including subsections of the current section
- **Shift+↑** Up
  - Go to parent section of the current section
- **Shift+↓** Down
  - Go to next section, skipping subsections of the current section

Plus in addition, the user agent could map the j and k keys to navigating to the previous or next element of heading content, regardless of the section's outline depth and ignoring sections with no headings.

4.3.12 Usage summary

This section is non-normative.

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>body[179]</td>
<td>The contents of the document.</td>
</tr>
<tr>
<td>article[183]</td>
<td>A complete, or self-contained, composition in a document, page, application, or site and that is, in principle, independently distributable or reusable, e.g. in syndication. This could be a forum post, a magazine or newspaper article, a blog entry, a user-submitted comment, an interactive widget or gadget, or any other independent item of content.</td>
</tr>
<tr>
<td>section[183]</td>
<td>A generic section of a document or application. A section, in this context, is a thematic grouping of content, typically with a heading.</td>
</tr>
<tr>
<td>Element</td>
<td>Purpose</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>nav</td>
<td>A section of a page that links to other pages or to parts within the page: a section with navigation links.</td>
</tr>
<tr>
<td>aside</td>
<td>A section of a page that consists of content that is tangentially related to the content around the aside element, and which could be considered separate from that content. Such sections are often represented as sidebars in printed typography.</td>
</tr>
<tr>
<td>h1-h6</td>
<td>A section heading</td>
</tr>
<tr>
<td>hgroup</td>
<td>The heading of a section, which consists of all the h1-h6 element children of the hgroup element. The element is used to group a set of h1-h6 elements when the heading has multiple levels, such as subheadings, alternative titles, or taglines.</td>
</tr>
<tr>
<td>header</td>
<td>A group of introductory or navigational aids.</td>
</tr>
<tr>
<td>footer</td>
<td>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.</td>
</tr>
</tbody>
</table>
4.3.12.1 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 reddit comment 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 an article because that was the whole thing.

A comment on an article is not part of the article on which it is commenting, therefore it is its own 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 can be omitted if the p element is immediately followed by an address, article, aside, blockquote, details, div, figcaption, figure, footer, form, h1, h2, h3, h4, h5, h6, header, hgroup, 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, or an autonomous custom element.

Content attributes:
- Global attributes

Accessibility considerations:
- For authors
- For implementers

DOM interface:

```
[Exposed=Window]
interface HTMLParagraphElement : HTMLElement {
    [HTMLConstructor] constructor();
    // also has obsolete members
};
```

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
The following examples are conforming HTML fragments:

```html
<p>The little kitten gently seated herself on a piece of carpet. Later in her 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, Whose markup was of dubious quality. The validator complained, So the author was pained, 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**

The following example is technically correct:

```html
<section>
<!-- ... -->
<p>Last modified: 2001-04-23</p>
<p>Author: fred@example.com</p>
</section>

However, it would be better marked-up as:

```html
<section>
<!-- ... -->
<footer>Last modified: 2001-04-23</footer>
<address>Author: fred@example.com</address>
</section>

Or:

```html
<section>
<!-- ... -->
<footer>
<p>Last modified: 2001-04-23</p>
<address>Author: fred@example.com</address>
</footer>
</section>
```

**Note**

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**
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

The markup for the above example could therefore be:

```html
<p>For instance, this fantastic sentence has bullets relating to</p>
<ul>
  <li>wizards,</li>
  <li>faster-than-light travel, and</li>
  <li>telepathy,</li>
</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

Thus for instance the above example could become the following:

```html
<div>For instance, this fantastic sentence has bullets relating to
<ul>
  <li>wizards,</li>
  <li>faster-than-light travel, and</li>
  <li>telepathy,</li>
</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.
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**

The following fictional extract from a project manual shows two sections that use the `hr` element to separate topics within the section.

```html
<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.
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>Maybe it won't be that bad, he told himself. The lie was comforting enough to get him through the rest of the night's shift.</p>

<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.

**Accessibility considerations**: For authors, For implementers.

**DOM interface**:

```javascript
[Exposed=Window]
interface HTMLPreElement : HTMLElement {
    [HTMLConstructor] constructor();
    // also has obsolete members
};
```

The `<pre>` element represents a block of preformatted text, in which structure is represented by typographic conventions rather than by elements.
Some examples of cases where the `pre` element could be used:

- Including an email, 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.

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.

Example

In the following snippet, a sample of computer code is presented.

```html
<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>
</code></code>
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 or more information about the edit

Accessibility considerations:
- For authors.
- For implementers.

DOM interface:

```idl
[Exposed=Window]
interface HTMLQuoteElement : HTMLElement {
    [HTMLConstructor] constructor();
    [CEReactions] attribute USVString cite;
};
```

Note: The `HTMLQuoteElement` interface is also used by the `q` element.

The `blockquote` element represents a section that is quoted from another source.

Content inside a `blockquote` must be quoted from another source, whose address, if it has one, may be cited in the `cite` attribute.

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 content of a `blockquote` may be abbreviated or may have context added in the conventional manner for the text's language.

**Example**

For example, in English this is traditionally done using square brackets. Consider a page with the sentence "Jane ate the cracker. She then said she liked apples and fish."; it could be quoted as follows:
<blockquote>
<p>[Jane] then said she liked [...] fish.</p>
</blockquote>

Attribution for the quotation, if any, must be placed outside the `blockquote` element.

**Example**

For example, here the attribution is given in a paragraph after the quote:

```html
<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>
</blockquote>
<p>— Stephen Roberts</p>
```

The other examples below show other ways of showing attribution.

The `cite` IDL attribute must reflect the element's `cite` content attribute.

**Example**

Here a `blockquote` element is used in conjunction with a `figure` element and its `figcaption` to clearly relate a quote to its attribution (which is not part of the quote and therefore doesn't belong inside the `blockquote` itself):

```html
<figure>
<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>
...</p>
</blockquote>
</figure>
```

**Example**

This next example shows the use of `cite` alongside `blockquote`.

```html
<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>
...</p>
</blockquote>
```

**Example**

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.

```html
<article>
<h1><a href="https://bacon.example.com/?blog=109431">Bacon on a crowbar</a></h1>
```
This example shows the use of a `blockquote` for short snippets, demonstrating that one does not have to use `p` elements inside `blockquote` elements:

```html
<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...</p>
```

**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` — Starting value of the list
  - `type` — Kind of list marker

**Accessibility considerations**
For authors.
For implementers.

**DOM interface**

```javascript
[Exposed=Window]
interface HTMLOLListElement : HTMLElement {
  [HTMLConstructor] constructor();

  [CFReactions] attribute boolean reversed;
  [CFReactions] attribute long start;
  [CFReactions] attribute DOMString type;

  // also has obsolete members
};
```

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. It is used to determine the starting value of the list.

An `ol` element has a starting value, which is an integer determined as follows:

1. If the `ol` element has a `start` attribute, then:
   1. Let parsed be the result of parsing the value of the attribute as an integer.
   2. If parsed is not an error, then return parsed.
2. If the `ol` element has a `reversed` attribute, then return the number of owned `li` elements.

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 identical to 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.
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.

For CSS user agents, a mapping for this attribute to the `list-style-type` CSS property is given in the rendering section (the mapping is straightforward: the states above have the same names as their corresponding CSS values).

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` and `type` IDL attributes must reflect the respective content attributes of the same name.

The `start` IDL attribute must reflect the content attribute of the same name, with a default value of 1.

This means that the `start` IDL attribute does not necessarily match the list's starting value, in cases where the `start` content attribute is omitted and the `reversed` content attribute is specified.

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.

```html
<p>I have lived in the following countries (given in the order of when I first lived there):</p>
<ol>
  <li>Switzerland</li>
  <li>United Kingdom</li>
  <li>United States</li>
  <li>Norway</li>
</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:

```html
<p>I have lived in the following countries (given in the order of when I first lived there):</p>
<ol>
  <li>United Kingdom</li>
  <li>Switzerland</li>
  <li>United States</li>
  <li>Norway</li>
</ol>
```

4.4.6 The `ul` element

Categories:

- Flow content.

If the element's children include at least one `li` element: Palpable content.
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**

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>
  <li>Switzerland</li>
  <li>United Kingdom</li>
  <li>United States</li>
</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>
  <li>Norway</li>
  <li>United Kingdom</li>
  <li>United States</li>
</ul>
```
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

Accessibility considerations:
For authors.
For implementers.

DOM interface:

```
[Exposed=Window]
interface HTMLMenuElement : HTMLElement {
    [HTMLConstructor] constructor();
    // also has obsolete members
}
```

The menu element represents a toolbar consisting of its contents, in the form of an unordered list of items (represented by li elements), each of which represents a command that the user can perform or activate.

**Note**
The menu element is simply a semantic alternative to ul to express an unordered list of commands (a "toolbar").

**Example**
In this example, a text-editing application uses a menu element to provide a series of editing commands:

```
<menu>
    <li><button onclick="copy()"><img src="copy.svg" alt="Copy"></button></li>
    <li><button onclick="cut()"><img src="cut.svg" alt="Cut"></button></li>
    <li><button onclick="paste()"><img src="paste.svg" alt="Paste"></button></li>
</menu>
```

Note that the styling to make this look like a conventional toolbar menu is up to the application.

4.4.8 The li element

Categories:
None.

Contexts in which this element can be used:
Inside ol elements.
Inside ul elements.
Inside menu elements.

Content model:
Flow content.

Tag omission in text/html:
An li element's end tag can 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

If the element is not a child of an `ul` or `menu` element: `value` — Ordinal value of the list item

Accessibility considerations

For authors.
For implementers.

DOM interface:

```javascript
[Exposed=Window]
interface HTMLLIElement : HTMLElement {
    [HTMLConstructor] constructor();

    [CFReactions] attribute long value;
    // also has obsolete members
};
```

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.

The `value` attribute, if present, must be a valid integer. It is used to determine the ordinal value of the list item, when the `li`'s list owner is an `ol` element.

Any element whose computed value of 'display' is 'list-item' has a list owner, which is determined as follows:

1. If the element is not being rendered, return null; the element has no list owner.
2. Let ancestor be the element’s parent.
3. If the element has an `ol`, `ul`, or `menu` ancestor, set ancestor to the closest such ancestor element.
4. Return the closest inclusive ancestor of ancestor that produces a CSS box.

Note: Such an element will always exist, as at the very least the document element will always produce a CSS box.

To determine the ordinal value of each element owned by a given list owner owner, perform the following steps:

1. Let i be 1.
2. If owner is an `ol` element, let numbering be owner’s starting value. Otherwise, let numbering be 1.
3. Loop: If i is greater than the number of list items that owner owns, then return; all of owner's owned list items have been assigned ordinal values.
4. Let item be the ith of owner’s owned list items, in tree order.
5. If item is an `li` element that has a value attribute, then:
   1. Let parsed be the result of parsing the value of the attribute as an integer.
   2. If parsed is not an error, then set numbering to parsed.
6. The ordinal value of item is numbering.
7. If owner is an `ol` element, and owner has a reversed attribute, decrement numbering by 1; otherwise, increment numbering by 1.
8. Increment i by 1.
9. Go to the step labeled loop.
The value IDL attribute must reflect the value of the value content attribute. 

**Example**

The element's value IDL attribute does not directly correspond to its ordinal value; it simply reflects the content attribute. For example, given this list:

```html
<ol>
  <li>Item 1</li>
  <li value="3">Item 3</li>
  <li>Item 4</li>
</ol>
```

The ordinal values are 1, 3, and 4, whereas the value IDL attributes return 0, 3, 0 on getting.

**Example**

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.

```html
<figure>
  <figcaption>The top 10 movies of all time</figcaption>
  <ol reversed="true">
    <li><cite>Josie and the Pussycats</cite>, 2001</li>
    <li lang="sh">Црна мачка, бели мачор</li>
    <li>A Bug's Life</li>
    <li>Toy Story</li>
    <li>Monsters, Inc</li>
    <li>Cars</li>
    <li>Toy Story 2</li>
    <li>Finding Nemo</li>
    <li>The Incredibles</li>
    <li>Ratatouille</li>
  </ol>
</figure>
```

The markup could also be written as follows, using the reversed attribute on the ol element:

```html
<figure>
  <figcaption>The top 10 movies of all time</figcaption>
  <ol reversed="true">
    <li><cite>Josie and the Pussycats</cite>, 2001</li>
    <li lang="sh">Црна мачка, бели мачор</li>
    <li>A Bug's Life</li>
    <li>Toy Story</li>
    <li>Monsters, Inc</li>
    <li>Cars</li>
    <li>Toy Story 2</li>
    <li>Finding Nemo</li>
    <li>The Incredibles</li>
    <li>Ratatouille</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.
4.4.9 The *dl* element

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:

Either: 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.

Or: One or more *div* elements, optionally intermixed with script-supporting elements.

Tag omission in text/html:

Neither tag is omissible.

Content attributes:

Global attributes

Accessibility considerations:

For authors.

For implementers.

DOM interface:

```idl
[Exposed=Window]
interface HTMLDListElement : HTMLElement {
  [HTMLConstructor] constructor();

  // also has obsolete members
};
```

The *dl* element represents an association list consisting of zero or more name-value groups (a description list). A name-value group consists of one or more names (*dt* elements, possibly as children of a *div* element child) followed by one or more values (*dd* elements, possibly as children of a *div* element child), ignoring any nodes other than *dt* and *dd* elements that are children of *div* element children. Within a single *dl* element, there should not be more than one *dt* element for each name.

Name-value groups may be terms and definitions, metadata topics and values, questions and answers, or any other groups of name-value data.

The values within a group are alternatives; multiple paragraphs forming part of the same value must all be given within the same *dd* element.

The order of the list of groups, and of the names and values within each group, may be significant.

In order to annotate groups with microdata attributes, or other global attributes that apply to whole groups, or just for styling purposes, each group in a *dl* element can be wrapped in a *div* element. This does not change the semantics of the *dl* element.

The name-value groups of a *dl* element are determined using the following algorithm. A name-value group has a name (a list of *dt* elements, initially empty) and a value (a list of *dd* elements, initially empty).

1. Let *groups* be an empty list of name-value groups.
2. Let *current* be a new name-value group.
3. Let *seenDd* be false.
4. Let *child* be *dl*’s first child.
5. Let *grandchild* be null.
6. While *child* is not null:
   1. If *child* is a *div* element, then:
1. Let grandchild be child's first child.

2. While grandchild is not null:
   1. Process dt or dd for grandchild.
   2. Set grandchild to grandchild's next sibling.

3. Otherwise, process dt or dd for child.

4. Set child to child's next sibling.

5. If current is not empty, then append current to groups.

6. Return groups.

To process dt or dd for a node node means to follow these steps:

1. Let groups, current, and seenDd be the same variables as those of the same name in the algorithm that invoked these steps.

2. If node is a dt element, then:
   1. If seenDd is true, then append current to groups, set current to a new name-value group, and set seenDd to false.
   2. Append node to current's name.

3. Otherwise, if node is a dd element, then append node to current's value and set seenDd to true.

Note When a name-value group has an empty list as name or value, it is often due to accidentally using dd elements in the place of dt elements and vice versa. Conformance checkers can spot such mistakes and might be able to advise authors how to correctly use the markup.

Example In the following example, one entry ("Authors") is linked to two values ("John" and "Luke").

```xml
<dl>
  <dt>Authors</dt>
  <dd>John</dd>
  <dd>Luke</dd>
  <dt>Editor</dt>
  <dd>Frank</dd>
</dl>
```

Example In the following example, one definition is linked to two terms.

```xml
<dl>
  <dt lang="en-US">color</dt>
  <dfn>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.</dfn>
</dl>
```

Example The following example illustrates the use of the dt element to mark up metadata of sorts. At the end of the example, one group has two metadata labels ("Authors" and "Editors") and two values ("Robert Rothman" and "Daniel Jackson"). This example also uses the div element around the groups of dt and dd element, to aid with styling.

```xml
<dl>
  <div>
```

```xml
</div>
```
Example

The following example shows the `<dl>` element used to give a set of instructions. The order of the instructions here is important (in the other examples, the order of the blocks was not important).

```html
<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>
```

Example

The following snippet shows a `<dl>` element being used as a glossary. Note the use of `<dfn>` to indicate the word being defined.

```html
<dl>
  <dt><dfn>Apartment</dfn>, n.</dt>
  <dd>An execution context grouping one or more threads with one or more COM objects.</dd>
  <dt><dfn>Flat</dfn>, n.</dt>
  <dd>A deflated tire.</dd>
  <dt><dfn>Home</dfn>, n.</dt>
  <dd>The user's login directory.</dd>
</dl>
```

Example

This example uses `microdata` attributes in a `<dl>` element, together with the `<div>` element, to annotate the ice cream desserts at a French restaurant.

```html
<dl>
  <div itemscope itemtype="http://schema.org/Product">
    <dt itemprop="name">Café ou Chocolat Liégeois</dt>
    <dd itemprop="offers" itemscope itemtype="http://schema.org/Offer">
      <span itemprop="price">3.50</span>
      <data itemprop="priceCurrency" value="EUR">€</data>
    </dd>
    <dd itemprop="description">
```
2 boules Café ou Chocolat, 1 boule Vanille, sause café ou chocolat, chantilly
</div>

<div itemscope itemtype="http://schema.org/Product">
  <dt itemprop="name">Américaine</dt>
  <dd itemprop="offers" itemscope itemtype="http://schema.org/Offer">
    <span itemprop="price">3.50</span>
  </dd>
  <dd itemprop="description">
    1 boule Crème brûlée, 1 boule Vanille, 1 boule Caramel, chantilly
  </dd>
</div>

Without the `div` element the markup would need to use the `itemref` attribute to link the data in the `dd` elements with the item, as follows.

```html
<dl>
  <dt itemscope itemtype="http://schema.org/Product" itemref="1-offer 1-description">
    <span itemprop="name">Café ou Chocolat Liégeois</span>
    <dd id="1-offer" itemprop="offers" itemscope itemtype="http://schema.org/Offer">
      <span itemprop="price">3.50</span>
      <data itemprop="priceCurrency" value="EUR">€</data>
    </dd>
    <dd id="1-description" itemprop="description">
      2 boules Café ou Chocolat, 1 boule Vanille, sause café ou chocolat, chantilly
    </dd>
  </dt>
  <dt itemscope itemtype="http://schema.org/Product" itemref="2-offer 2-description">
    <span itemprop="name">Américaine</span>
    <dd id="2-offer" itemprop="offers" itemscope itemtype="http://schema.org/Offer">
      <span itemprop="price">3.50</span>
      <data itemprop="priceCurrency" value="EUR">€</data>
    </dd>
    <dd id="2-description" itemprop="description">
      1 boule Crème brûlée, 1 boule Vanille, 1 boule Caramel, chantilly
    </dd>
  </dt>
</dl>
```

Note
---

The `dl` element is inappropriate for marking up dialogue. See some examples of how to mark up dialogue.

4.4.10 The `dt` element

**Categories**

None.

**Contexts in which this element can be used**

- Before `dd` or `dt` elements inside `dl` elements.
- Before `dd` or `dt` elements inside `div` elements that are children of a `dl` element.

**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` can be omitted if the `dt` element is immediately followed by another `dt` element or a `dd` element.

**Content attributes**

- Global attributes

**Accessibility considerations**

For authors.
The `dt` element represents the term, or name, part of a term-description group in a description list (`dl` element).

**Note**
The `dt` element itself, when used in a `dl` element, does not indicate that its contents are a term being defined, but this can be indicated using the `dfn` element.

**Example**
This example shows a list of frequently asked questions (a FAQ) marked up using the `dt` element for questions and the `dd` element for answers.

```html
<article>
  <h1>FAQ</h1>
  <dl>
    <dt>What do we want?</dt>
    <dd>Our data.</dd>
    <dt>When do we want it?</dt>
    <dd>Now.</dd>
    <dt>Where is it?</dt>
    <dd>We are not sure.</dd>
  </dl>
</article>
```

### 4.4.11 The aa element

**Categories**

None.

**Contexts in which this element can be used**

- After `dt` or `dd` elements inside `dl` elements.
- After `dt` or `dd` elements inside `div` elements that are children of a `dl` element.

**Content model**

Flow content.

**Tag omission in text/html**

A `dd` element's end tag can 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.

**Accessibility considerations**

- For authors.
- For implementers.

**DOM interface**

Uses `HTMLElement`.

The `dd` element represents the description, definition, or value, part of a term-description group in a description list (`dl` element).

**Example**
A `dd` can be used to define a vocabulary list, like in a dictionary. In the following example, each entry, given by a `dt` element with a `dfn` element, has several `dd`s, showing the various parts of the definition.
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:
- Either: one `figcaption` element followed by flow content.
- Or: flow content followed by one `figcaption` element.
- Or: flow content.

### Tag omission in text/html:
Neither tag is omissible.

### Content attributes:
- Global attributes

### Accessibility considerations:
- For authors.
- For implementers.

### DOM interface:
Uses `HTMLElement`.

The `figure` element optionally with a caption, that is self-contained (like a complete sentence) and is typically 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.

**Note**

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 first `figcaption` element child of the element, 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

This example shows the `figure` element to mark up a code listing.

```html
<p>In <a href="#l4">listing 4</a> 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();
    undefined sendData(sequence&lt;byte&gt; data);
    undefined initSelfDestruct();
}</code></pre>
</figure>
<p>The API is designed to use UTF-8.</p>
```

Example

Here we see a `figure` element to mark up a photo that is the main content of the page (as in a gallery).

```html
<!DOCTYPE HTML>
<html lang="en">
<title>Bubbles at work — My Gallery™</title>
<figure>
  <img src="bubbles-work.jpeg" alt="Bubbles, sitting in his office chair, works on his latest project intently.">
  <figcaption>Bubbles at work</figcaption>
</figure>
```

Example

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.

```html
<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:

```
</h2>
```

Example

```
<blockquote>
  <img src="promblem-packed-action.png" alt="ROUGH COPY! Promblem-Packed Action!">
</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.

```

Example

```
<figure>
  <img src="ex-a.png" alt="Two squiggles on a dirty piece of paper.">
</figure>
```
Exhibit A. The alleged comic.

Exhibit B. The trailer.

The case was resolved out of court.

Example

Here, a part of a poem is marked up using figure.

'Twas brillig, and the slithy toves Did gyre and gimble in the wabe;
All mimsy were the borogoves, And the mome raths outgrabe.

(first verse). Lewis Carroll, 1832-98

Example

In this example, which could be part of a much larger work discussing a castle, nested elements are used to provide both a group caption and individual captions for each figure in the group:

The castle through the ages: 1423, 1858, and 1999 respectively.

Example

The previous example could also be more succinctly written as follows (using attributes in place of the nested pairs):
4.4.13 The figcaption element

Categories:
None.

Contexts in which this element can be used:
As the first or last child of a figure element.

Content model:
Flow content.

Tag omission in text/html:
Neither tag is omissible.

Content attributes:
Global attributes.

Accessibility considerations:
For authors.
For implementers.

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.

Example
The element can contain additional information about the source:

```
<figcaption>
<p>A duck.</p>
<p>Photograph courtesy of 🌟 News.</p>
</figcaption>

<figcaption>
<p>Average rent for 3-room apartments, excluding non-profit apartments.</p>
</figcaption>
```
4.4.14 The `main` element

Categories:
- Flow content
- Palpable content

Contexts in which this element can be used:
Where flow content, is expected, but only if it is a hierarchically correct `main` element.

Content model:
- Flow content

Tag omission in text/html:
Neither tag is omissible.

Content attributes:
- Global attributes

Accessibility considerations:
- For authors.
- For implementers.

DOM interface:
Uses `HTMLElement`.

The `main` element represents the dominant contents of the document.

A document must not have more than one `main` element that does not have the `hidden` attribute specified.

A hierarchically correct `main` element is one whose ancestor elements are limited to `html`, `body`, `div`, `form` without an accessible name, and autonomous custom elements. Each `main` element must be a hierarchically correct `main` element.

Example
In this example, the author has used a presentation where each component of the page is rendered in a box. To wrap the main content of the page (as opposed to the header, the footer, the navigation bar, and a sidebar), the `main` element is used.

```html
<!DOCTYPE html>
<html lang="en">
<title>RPG System 17</title>
<style>
header, nav, aside, main, footer {
    margin: 0.5em; border: thin solid; padding: 0.5em;
    background: #EFF; color: black; box-shadow: 0 0 0.25em #033;
}

h1, h2, p { margin: 0; }
nav, main { float: left; }
aside { float: right; }
footer { clear: both; }
</style>
<header>
<h1>System Eighteen</h1>
</header>
<nav>
<a href="../16/">← System 17</a>
<a href="../18/">RPXIX →</a>
</nav>
<aside>
<p>This system has no HP mechanic, so there's no healing.</p>
</aside>
<main>
<h2>Character creation</h2>
<p>Attributes (magic, strength, agility) are purchased at the cost of one point per level.</p>
</main>
</html>
```
In the following example, multiple `<main>` elements are used and script is used to make navigation work without a server roundtrip and to set the `<hidden>` attribute on those that are not current:

```html
<!doctype html>
<html lang=en-CA>
<meta charset=utf-8>
<title>…</title>
<link rel=stylesheet href=spa.css>
<script src=spa.js async></script>
<nav>
  <a href=/Home/>Home</a>
  <a href=/about>About</a>
  <a href=/contact>Contact</a>
</nav>
<main>
  <h1>Home</h1>
  …
</main>
<main hidden>
  <h1>About</h1>
  …
</main>
<main hidden>
  <h1>Contact</h1>
  …
</main>
<footer>Made with ❤ by <a href=https://example.com/>Example 🎃</a>.</footer>
```

---

### 4.4.15 The `<div>` element

**Categories:**
- Flow content
- Paipable content

**Contexts in which this element can be used:**
Where flow content is expected.
As a child of a `<dl>` element.

**Content model:**
If the element is a child of a `<dl>` element: one or more `<dt>` elements followed by one or more `<dd>` elements, optionally intermixed with script-supporting elements.
If the element is not a child of a `<dl>` element: flow content.

**Tag omission in text/html:**
Neither tag is omissible.

**Content attributes:**
- Global attributes

**Accessibility considerations:***
For authors.
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. It can also be used in a `dl` element, wrapping groups of `dt` and `dd` 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
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:

```html
<article lang="en-US">
  <h1>My use of language and my cats</h1>
  <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, coloured 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 recognises that their flat is a mirror image of ours.</p>
  </div>
  <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 "colour"
</p>
</article>
```

4.5 Text-level semantics

4.5.1 The `a` element

Categories:
- Flow content
- Phrasing content
- 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 descendant, a element descendant, or descendant with the attribute specified.

Tag omission in text/html:

- Neither tag is omissible.

Content attributes:

- Global attributes:
  - href — Address of the hyperlink
  - target — Browsing context for hyperlink navigation
  - download — Whether to download the resource instead of navigating to it, and its file name if so
  - ping — Relationship between the location in the document containing the hyperlink and the destination resource
  - rel — Language of the linked resource
  - hreflang — Hint for the type of the referenced resource
  - referrerpolicy — Referrer policy for fetches initiated by the element

Accessibility considerations:

- If the element has an href attribute: for authors; for implementers.
- Otherwise: for authors; for implementers.

DOM interface:

```idl
[Exposed=Window]
interface HTMLAnchorElement : HTMLElement {

[HTMLConstructor] constructor();

[CEReactions] attribute DOMString target;
[CEReactions] attribute DOMString download;
[CEReactions] attribute USVString ping;
[CEReactions] attribute DOMString rel;
[SameObject, PutForwards=value] readonly attribute DOMTokenList relList;
[CEReactions] attribute DOMString hreflang;
[CEReactions] attribute DOMString type;

[CEReactions] attribute DOMString text;

[CEReactions] attribute DOMString referrerPolicy;

// also has obsolete members
}

HTMLAnchorElement includes HTMLHyperlinkElementUtils;
```

If the element has an href attribute, then it represents a hyperlink (a hypertext anchor) labeled by its contents.

If the 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, ping, rel, hreflang, type, and referrerpolicy attributes must be omitted if the href attribute is not present.

If the itemprop attribute is specified on an element, then the href attribute must also be specified.

Example:

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 element:

```html
<nav>
  <ul>
    <li> <a href="/">Home</a> </li>
    <li> <a href="/news">News</a> </li>
  </ul>
</nav>
```
The `href`\(^{285}\), `target`\(^{285}\), `download`\(^{285}\), `ping`\(^{285}\), and `referrerpolicy`\(^{285}\) attributes affect what happens when users follow hyperlinks\(^{291}\), or download hyperlinks\(^{292}\), created using the `a`\(^{239}\) element. The `rel`\(^{285}\), `hreflang`\(^{285}\), and `type`\(^{285}\) 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`\(^{239}\) elements that create hyperlinks\(^{294}\) is to run the following steps:

1. If the target of the `click` event is an `img`\(^{321}\) element with an `ismap`\(^{324}\) attribute specified, then server-side image map processing must be performed, as follows:
   1. Let `x` and `y` be zero.
   2. If the `click` event was a real pointing-device-triggered `click` event on the `img`\(^{321}\) element, then set `x` to the distance in CSS pixels from the left edge of the image to the location of the click, and set `y` to the distance in CSS pixels from the top edge of the image to the location of the click.
   3. If `x` is negative, set `x` to zero.
   4. If `y` is negative, set `y` to zero.
   5. 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.

2. Follow the hyperlink\(^{291}\) or download the hyperlink\(^{292}\) created by the `a`\(^{239}\) element, as determined by the `download`\(^{285}\) attribute and any expressed user preference, passing `hyperlink suffix`, if the steps above defined it.

**For web developers (non-normative)**

```
.a.text\(^{241}\)

Same as `textContent`.  
```

The IDL attributes `download`, `ping`, `target`, `rel`, `hreflang`, and `type`, must `reflect`\(^{295}\) the respective content attributes of the same name.

The IDL attribute `relList` must `reflect`\(^{295}\) the `rel`\(^{285}\) content attribute.

The IDL attribute `referrerPolicy` must `reflect`\(^{295}\) the `referrerPolicy`\(^{285}\) content attribute, limited to only known values\(^{295}\).

The `text` attribute's getter must return this element's descendant text content.

The `text`\(^{241}\) attribute's setter must `string replace all` with the given value within this element.

**Example**

The `a`\(^{239}\) element can 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:

```html
<aside class="advertising">
  <h1>Advertising</h1>
  <a href="https://ad.example.com/?adid=1929&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&pubid=1422">
    <section>
      <!-- Advertising content here... -->
    </section>
  </a>
</aside>
```
The following example shows how a bit of script can be used to effectively make an entire row in a job listing table a hyperlink:

```html
<table>
  <tr>
    <th>Position</th>
    <th>Team</th>
    <th>Location</th>
  </tr>
  <tr>
    <td><a href="/jobs/manager">Manager</a></td>
    <td>Remotees</td>
    <td>Remote</td>
  </tr>
  <tr>
    <td><a href="/jobs/director">Director</a></td>
    <td>Remotees</td>
    <td>Remote</td>
  </tr>
  <tr>
    <td><a href="/jobs/astronaut">Astronaut</a></td>
    <td>Architecture</td>
    <td>Remote</td>
  </tr>
</table>
<script>
  document.querySelector("table").onclick = ( { target } ) => {
    if (target.parentElement.localName === "tr") {
      const link = target.parentElement.querySelector("a");
      if (link) {
        link.click();
      }
    }
  }
</script>
```
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**

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 cute</em> animals! </p>
```

**Note**

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

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

Accessibility considerations:
For authors.
For implementers.

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. (This is distinct from marking up subheadings, for which the **hgroup** element is appropriate.)

**Example**

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**

Here, the word "chapter" and the actual chapter number are mere boilerplate, and the actual name of the chapter is marked up with **strong**:

```html
<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):

```html
<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.

```html
<h1><strong>Flowers, Bees, and Honey</strong> and other things I don't understand</h1>
```

**Example**

Here is an example of a warning notice in a game, with the various parts marked up according to how important they are:
Example

In this example, the `strong` element is used to denote the part of the text that the user is intended to read first.

```
<em>Warning.</em> This dungeon is dangerous.
<em>Avoid the ducks.</em> Take any gold you find.
<em>Do not take any of the diamonds</em>, they are explosive and <em>will destroy anything within ten meters.</em> You have been warned.
```

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**

Accessibility considerations:
- For authors.
- For implementers.

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.
The `small` element must not be used for subheadings; for that purpose, use the `hgroup` element.

**Example**

In this example, the `small` element is used to indicate that value-added tax is not included in a price of a hotel room:

```html
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**

In this second example, the `small` element is used for a side comment in an article.

```html
<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.

```html
<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>
```

**Example**

In this last example, the `small` element is marked as being important small print.

```html
<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.
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**

In this example a recommended retail price has been marked as no longer relevant as the product in question has a new sale price.

```html
<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

**Accessibility considerations**
- For authors
- For implementers

**DOM interface**
Uses `HTMLElement`.

The **`cite`** element **represents** the title of a work (e.g. 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, etc). This can be a work that is being quoted or referenced in detail (i.e. a citation), or it can just be a work that is mentioned in passing.

A person's name is not the title of a work — even if people call that person a piece of work — and the element must therefore not be used to mark up people's names. (In some cases, the `b` element might be appropriate for names; e.g. in a gossip article where the names of famous people are keywords rendered with a different style to draw attention to them. In other cases, if an element is really needed, the `span` element can be used.)
This next example shows a typical use of the `cite` element:

```html
<p>My favorite book is <cite>The Reality Dysfunction</cite> by Peter F. Hamilton. My favorite comic is <cite>Pearls Before Swine</cite> by Stephan Pastis. My favorite track is <cite>Jive Samba</cite> by the Cannonball Adderley Sextet.</p>
```

This is correct usage:

```html
<p>According to the Wikipedia article <cite>HTML</cite>, as it stood in mid-February 2008, leaving attribute values unquoted is unsafe. This is obviously an over-simplification.</p>
```

The following, however, is incorrect usage, as the `cite` element here is containing far more than the title of the work:

```html
<!-- do not copy this example, it is an example of bad usage! -->
<p>According to <cite>the Wikipedia article on HTML</cite>, as it stood in mid-February 2008, leaving attribute values unquoted is unsafe. This is obviously an over-simplification.</p>
```

The `cite` element is obviously a key part of any citation in a bibliography, but it is only used to mark the title:

```html
```

A citation is <i>not</i> a quote (for which the `q` element is appropriate).

This is incorrect usage, because `cite` is not for quotes:

```html
<p><cite>This is wrong!</cite>, said Ian.</p>
```

This is also incorrect usage, because a person is not a work:

```html
<p><q>This is still wrong!</q>, said <cite>Ian</cite>.</p>
```

The correct usage does not use a `cite` element:

```html
<p><q>This is correct</q>, said Ian.</p>
```

As mentioned above, the `b` element might be relevant for marking names as being keywords in certain kinds of documents:

```html
<p>And then <b>Ian</b> said <q>this might be right, in a gossip column, maybe!</q>.</p>
```

4.5.7 The `q` element

Categories: Flow content

Flow content:
Phrasing content\textsuperscript{p133}

Palpable content\textsuperscript{p134}

Contexts in which this element can be used\textsuperscript{p129}:
Where phrasing content\textsuperscript{p133} is expected.

Content model\textsuperscript{p129}:
Phrasing content\textsuperscript{p133}

Tag omission in text/html\textsuperscript{p130}:
Neither tag is omissible.

Content attributes\textsuperscript{p130}:
Global attributes\textsuperscript{p137}

cite\textsuperscript{p249} — Link to the source of the quotation or more information about the edit

Accessibility considerations\textsuperscript{p130}:
For authors.
For implementers.

DOM interface\textsuperscript{p130}:
Uses HTMLQuoteElement\textsuperscript{p219}

The \texttt{q} element \textit{represents} some phrasing content\textsuperscript{p133} 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 \texttt{q} elements; they will be inserted into the rendering by the user agent.

Content inside a \texttt{q} element must be quoted from another source, whose address, if it has one, may be cited in the \texttt{cite} attribute. The source may be fictional, as when quoting characters in a novel or screenplay.

If the \texttt{cite} attribute is present, it must be a valid URL potentially surrounded by spaces.\textsuperscript{p88} To obtain the corresponding citation link, the value of the attribute must be \texttt{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 \texttt{q} element must not be used in place of quotation marks that do not represent quotes; for example, it is inappropriate to use the \texttt{q} element for marking up sarcastic statements.

The use of \texttt{q} elements to mark up quotations is entirely optional; using explicit quotation punctuation without \texttt{q} elements is just as correct.

Example

Here is a simple example of the use of the \texttt{q} element:

```html
<p>The man said <q>Things that are impossible just take longer</q>. I disagreed with him.</p>
```

Example

Here is an example with both an explicit citation link in the \texttt{q} element, and an explicit citation outside:

```html
<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

In the following example, the quotation itself contains a quotation:

```html
<p>In <cite>Example One</cite>, he writes <q>The man</q>
```
said <q>Things that are impossible just take longer</q>. I disagreed with him</q>. Well, I disagree even more!</p>

**Example**

In the following example, quotation marks are used instead of the `<q>` element:

```html
<p>His best argument was “I disagree”, which I thought was laughable.</p>
```

**Example**

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.

```html
<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: Full term or expansion of abbreviation.

**Accessibility considerations**:
- For authors
- For implementers

**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 `descendant text content` 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 \texttt{a} element that links to a \texttt{dfn} element represents an instance of the term defined by the \texttt{dfn} element.

**Example**

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 \texttt{<dfn><abbr title="Garage Door Opener">GDO</abbr></dfn>}
is a device that allows off-world teams to open the iris.</p>
<p>Teal'c activated his \texttt{<abbr title="Garage Door Opener">GDO</abbr>}
and so Hammond ordered the iris to be opened.</p>
```

With the addition of an \texttt{a} element, the \texttt{reference} can be made explicit:

```
<p>The \texttt{<dfn id=gdo><abbr title="Garage Door Opener">GDO</abbr></dfn>}
is a device that allows off-world teams to open the iris.</p>
<p>Teal'c activated his \texttt{<a href=gdo><abbr title="Garage Door Opener">GDO</abbr></a>}
and so Hammond ordered the iris to be opened.</p>
```

```
Example
```

The paragraph below contains an abbreviation marked up with the \texttt{abbr} element. This paragraph \texttt{defines the term} “Web Hypertext Application Technology Working Group”.

```
<p>The \texttt{<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
```

4.5.9 The \texttt{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
  - The \texttt{title} attribute has special semantics on this element: Full term or expansion of abbreviation.

Accessibility considerations:
- For authors
- For implementers

DOM interface:
- Uses HTMLElement

The \texttt{abbr} element represents an abbreviation or acronym, optionally with its expansion. The \texttt{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.
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

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 HTML5 in 2004.</p>

Example

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

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

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

**Accessibility considerations**: For authors, For implementers.

**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*.

The content model of `ruby` elements consists of one or more of the following sequences:

1. One or the other of the following:
   - Phrasing content, but with no `ruby` elements and with no `ruby` element descendants
   - A single `ruby` element that itself has no `ruby` element descendants

2. One or the other of the following:
   - One or more `rt` elements
   - An `rp` element followed by one or more `rt` elements, each of which is itself followed by an `rp` element

The `ruby` 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.

**Note**

At the time of writing, CSS does not yet provide a way to fully control the rendering of the HTML `ruby` element. It is hoped that CSS will be extended to support the styles described below in due course.

**Mono-ruby for individual base characters in Japanese**

One or more hiragana or katakana characters (the ruby annotation) are placed with each ideographic character (the base text). This is used to provide readings of kanji characters.
Mono-ruby for compound words (jukugo)

This is similar to the previous case: each ideographic character in the compound word (the base text) has its reading given in hiragana or katakana characters (the ruby annotation). The difference is that the base text segments form a compound word rather than being separate from each other.

Example

In this example, notice how each annotation corresponds to a single base character.

<ruby>君<rt>くん</rt><ruby>子<rt>し</rt>は<ruby>和<rt>わ</rt>して<ruby>同<rt>どう</rt>ぜず。</ruby></ruby>

君くん子は和わして同どうぜず。

This example can also be written as follows, using one ruby elements 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):

<ruby>君<rt>くん</rt>子<rt>し</rt>は<ruby>和<rt>わ</rt>して<ruby>同<rt>どう</rt>ぜず。</ruby></ruby>

Example

In this example, notice again how each annotation corresponds to a single base character. In this example, each compound word (jukugo) corresponds to a single ruby element.

The rendering here is expected to be that each annotation be placed over (or next to, in vertical text) the corresponding base character, with the annotations not overhanging any of the adjacent characters.

<ruby>鬼<rt>き</rt>門<rt>もん</rt><ruby>の<rt>ほう</rt>角<rt>がく</rt>を<ruby>凝<rt>ぎょう</rt>視<rt>し</rt>る</ruby></ruby>

鬼き門もんのほう角がくを凝ぎょう視しる

Jukugo-ruby

This is semantically identical to the previous case (each individual ideographic character in the base compound word has its reading given in an annotation in hiragana or katakana characters), but the rendering is the more complicated Jukugo Ruby rendering.

Example

This is the same example as above for mono-ruby for compound words. The different rendering is expected to be achieved using different styling (e.g. in CSS), and is not shown here.

<ruby>鬼<rt>き</rt>門<rt>もん</rt><ruby>の<rt>ほう</rt>角<rt>がく</rt>を<ruby>凝<rt>ぎょう</rt>視<rt>し</rt>る</ruby></ruby>

Note

For more details on Jukugo Ruby rendering, see Appendix F in the Requirements for Japanese Text Layout. [JLREQ]253

Group ruby for describing meanings

The annotation describes the meaning of the base text, rather than (or in addition to) the pronunciation. As such, both the base text and the annotation can be multiple characters long.

Example
**Group ruby for Jukuji readings**

A phonetic reading that corresponds to multiple base characters, because a one-to-one mapping would be difficult. (In English, the words "Colonel" and "Lieutenant" are examples of words where a direct mapping of pronunciation to individual letters is, in some dialects, rather unclear.)

**Example**

Here a compound ideographic word has its corresponding katakana given as an annotation.

<ruby><rb>境界面</rb><rt>インターフェース</rt></ruby>

境界面インターフェース

**Text with both phonetic and semantic annotations (double-sided ruby)**

Sometimes, ruby styles described above are combined.

If this results in two annotations covering the same single base segment, then the annotations can just be placed back to back.

**Example**

<ruby><rb>A</rb><rt>a</rt>�<rb>S</rb><rt>t</rt>�</ruby>

Example

In this contrived example, some symbols are given names in English and French.

<ruby>
  <rt lang="fr">Cœur</rt>
  <rt lang="fr">Trèfle</rt>
  <rt lang="fr">Étoile</rt>
</ruby>

In more complication situations such as following examples, a nested ruby element is used to give the inner annotations, and then that whole ruby is then given an annotation at the "outer" level.

**Example**

<ruby><ruby><rb>B</rb><rt>a</rt>�</ruby><rb>A</rb><rt>a</rt>�</ruby><ruby><rb>S</rb><rt>a</rt>�</ruby><ruby><rb>E</rb><rt>a</rt>�</ruby><ruby>annotation</ruby>

Example
Here both a phonetic reading and the meaning are given in ruby annotations. The annotation on the nested `ruby` element gives a mono-ruby phonetic annotation for each base character, while the annotation in the `rt` element that is a child of the outer `ruby` element gives the meaning using hiragana.

```xml
<ruby><ruby>東<rt>とう</rt>南<rt>なん</rt></ruby><rt>たつみ</rt></ruby>の方角
```

東とうなんたつみの方角

Example

This is the same example, but the meaning is given in English instead of Japanese:

```xml
<ruby><ruby>東<rt>とう</rt>南<rt>なん</rt></ruby><rt lang=en>Southeast</rt></ruby>の方角
```

東とうなんSoutheastの方角

Within a `ruby` element that does not have a `ruby` element ancestor, content is segmented and segments are placed into three categories: base text segments, annotation segments, and ignored segments. Ignored segments do not form part of the document's semantics (they consist 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). Base text segments can overlap (with a limit of two segments overlapping any one position in the DOM, and with any segment having an earlier start point than an overlapping segment also having an equal or later end point, and any segment have a later end point than an overlapping segment also having an equal or earlier start point). Annotation segments correspond to `rt` elements. Each annotation segment can be associated with a base text segment, and each base text segment can have annotation segments associated with it. (In a conforming document, each base text segment is associated with at least one annotation segment, and each annotation segment is associated with one base text segment.) A `ruby` element represents the union of the segments of base text it contains, along with the mapping from those base text segments to annotation segments. Segments are described in terms of DOM ranges; annotation segment ranges always consist of exactly one element. [DOM]

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 base text segments be an empty list of base text segments, each potentially with a list of base text subsegments.
2. Let annotation segments be an empty list of annotation segments, each potentially being associated with a base text segment or subsegment.
3. Let root be the `ruby` element for which the algorithm is being run.
4. If root has a `ruby` element ancestor, then jump to the step labeled `end`.
5. Let current parent be root.
6. Let index be 0.
7. Let start index be null.
8. Let parent start index be null.
9. Let current base text be null.
10. **Start mode:** If index is equal to or greater than the number of child nodes in current parent, then jump to the step labeled `end mode`.
11. If the indexth node in current parent is an `rt` or `rp` element, jump to the step labeled `annotation mode`.
12. Set start index to the value of index.
13. **Base mode:** If the indexth node in current parent is a `ruby` element, and if current parent is the same element as root, then push a ruby level, and then jump to the step labeled `start mode`.
14. If the indexth node in current parent is an `rt` or `rp` element, then set the current base text and then jump to the step labeled `annotation mode`.
15. Increment index by one.
16. **Base mode post-increment**: If `index` is equal to or greater than the number of child nodes in `current parent`, then jump to the step labeled **end mode**.

17. Jump back to the step labeled **base mode**.

18. **Annotation mode**: If the `index`th node in `current parent` is an \texttt{rt} element, then **push a ruby annotation** and jump to the step labeled **annotation mode increment**.

19. If the `index`th node in `current parent` is an \texttt{rp} element, jump to the step labeled **annotation mode increment**.

20. If the `index`th node in `current parent` is not a \texttt{Text} node, or is a \texttt{Text} node that is not **inter-element whitespace**, then jump to the step labeled **base mode**.

21. **Annotation mode increment**: Let `lookahead index` be `index` plus one.

22. **Annotation mode white-space skipper**: If `lookahead index` is equal to the number of child nodes in `current parent` then jump to the step labeled **end mode**.

23. If the `lookahead index`th node in `current parent` is an \texttt{rt} element or an \texttt{rp} element, then set `index` to `lookahead index` and jump to the step labeled **annotation mode**.

24. If the `lookahead index`th node in `current parent` is not a \texttt{Text} node, or is a \texttt{Text} node that is not **inter-element whitespace**, then jump to the step labeled **base mode** (without further incrementing `index`, so the **inter-element whitespace** seen so far becomes part of the next base text segment).

25. Increment `lookahead index` by one.

26. Jump to the step labeled **annotation mode white-space skipper**.

27. **End mode**: If `current parent` is not the same element as \texttt{root}, then **pop a ruby level** and jump to the step labeled **base mode post-increment**.

28. **End**: Return base text segments and annotation segments. Any content of the \texttt{ruby} element not described by segments in either of those lists is implicitly in an **ignored segment**.

When the steps above say to **set the current base text**, it means to run the following steps at that point in the algorithm:

1. Let `text range` be a DOM range whose **start** is the **boundary point** (`current parent`, `start index`) and whose **end** is the **boundary point** (`current parent`, `index`).
2. Let `new text segment` be a base text segment described by the range **annotation range**.
3. Add `new text segment` to **base text segments**.
4. Let **current base text** be `new text segment`.
5. Let `start index` be null.

When the steps above say to **push a ruby level**, it means to run the following steps at that point in the algorithm:

1. Let **current parent** be the `index`th node in `current parent`.
2. Let `index` be 0.
3. Set `saved start index` to the value of `start index`.
4. Let `start index` be null.

When the steps above say to **pop a ruby level**, it means to run the following steps at that point in the algorithm:

1. Let `index` be the position of `current parent` in `root`.
2. Let **current parent** be `root`.
3. Increment `index` by one.
4. Set `start index` to the value of `saved start index`.
5. Let `saved start index` be null.

When the steps above say to **push a ruby annotation**, it means to run the following steps at that point in the algorithm:
1. Let \( rt \) be the element that is the index\( \text{th} \) node of current parent.

2. Let annotation range be a DOM range whose start is the boundary point (current parent, index) and whose end is the boundary point (current parent, index plus one) (i.e. that contains only \( rt \)).

3. Let new annotation segment be an annotation segment described by the range annotation range.

4. If current base text is not null, associate new annotation segment with current base text.

5. Add new annotation segment to annotation segments.

Example

In this example, each ideograph in the Japanese text 漢字 is annotated with its reading in hiragana.

...<ruby>漢<rt>かん</rt>字<rt>じ</rt></ruby>
... This might be rendered as:

![kan ji]

... 漢字 ...

Example

In this example, each ideograph in the traditional Chinese text 漢字 is annotated with its bopomofo reading.

<ruby>漢<rt>ㄏㄢˋ</rt>字<rt>ㄗˋ</rt></ruby>

This might be rendered as:

![han zi]

Example

In this example, each ideograph in the simplified Chinese text 漢字 is annotated with its pinyin reading.

...<ruby>汉<rt>hàn</rt>字<rt>zì</rt></ruby>... This might be rendered as:

![han zi]

Example

In this more contrived example, the acronym "HTML" has four annotations: one for the whole acronym, briefly describing what it is, one for the letters "HT" expanding them to "Hypertext", one for the letter "M" expanding it to "Markup", and one for the letter "L" expanding it to "Language".
4.5.11 The **rt** 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 **rt** element’s end tag can 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.

**Content attributes**:

**Accessibility considerations**:

**For authors**.

**For implementers**.

**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, it doesn’t represent anything itself, but the **ruby** element uses it as part of determining what it represents.

An **rt** element that is not a child of a **ruby** element represents the same thing as its children.

4.5.12 The **rp** element

**Categories**: None.

**Contexts in which this element can be used**: As a child of a **ruby** element, either immediately before or immediately after an **rt** element.

**Content model**: Text

**Tag omission in text/html**: An **rp** element’s end tag can 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.

**Content attributes**:

**Accessibility considerations**:

**For authors**.

**For implementers**.

**DOM interface**: Uses **HTMLElement**.
The `rp` element can be used to provide parentheses or other content around a ruby text component of a ruby annotation, to be shown by user agents that don't 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.

**Example**

The example above, 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:

```html
...<ruby>漢<rp>( </rp><rt>かん</rt><rp> )</rp><rt>字</rt><rp>( </rp><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:

```html
...漢(かん)字(じ)... 
```

**Example**

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, but this time with `rp` elements as well:

```html
<ruby>
♥<rp>: </rp><rt>Heart</rt><rp>, </rp><rt lang=fr>Cœur</rt><rp>.</rp><rp></rp><rp></rp>
♠<rp>: </rp><rt>Shamrock</rt><rp>, </rp><rt lang=fr>Trèfle</rt><rp>.</rp><rp></rp><rp></rp>
♦<rp>: </rp><rt>Star</rt><rp>, </rp><rt lang=fr>Étoile</rt><rp>.</rp><rp></rp><rp></rp>
</ruby>
```

This would make the example render as follows in non-ruby-capable user agents:

```
```

4.5.13 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.

**Accessibility considerations**: For authors, For implementers.

**DOM interface**: 

```html
[Exposed=Window]
```
The `data`<sup>268</sup> element represents its contents, along with a machine-readable form of those contents in the `value`<sup>261</sup> 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`<sup>261</sup> element can be used instead.*

The element can be used for several purposes.

When combined with microformats or the microdata attributes<sup>271</sup> defined in this specification, 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.

**Example**

Here, a short table has its numeric values encoded using the `data`<sup>268</sup> element so that the table sorting JavaScript library can provide a sorting mechanism on each column despite the numbers being presented in textual form in one column and in a decomposed form in another.

```html
<script src="sortable.js"></script>
<table class="sortable">
<thead>
<tr><th>Game</th><th>Corporations</th><th>Map Size</th></tr>
</thead>
<tbody>
<tr><td>1830</td><td><data value="8">Eight</data></td><td><data value="93">19+74 hexes (93 total)</data></td></tr>
<tr><td>1856</td><td><data value="11">Eleven</data></td><td><data value="99">12+87 hexes (99 total)</data></td></tr>
<tr><td>1870</td><td><data value="10">Ten</data></td><td><data value="149">4+145 hexes (149 total)</data></td></tr>
</tbody>
</table>
```

### 4.5.14 The `time` element §<sup>26</sup>1

**Categories**<sup>129</sup>:

- Flow content<sup>132</sup>
- Phrasing content<sup>131</sup>
- Palpable content<sup>134</sup>

**Contexts in which this element can be used**<sup>129</sup>:

Where phrasing content<sup>133</sup> is expected.

**Content model**<sup>129</sup>:

If the element has a `datetime`<sup>262</sup> attribute: Phrasing content<sup>133</sup>. Otherwise: Text<sup>133</sup>, 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

Accessibility considerations:
For authors.
For implementers.

DOM interface:

```idl
[Exposed=Window]
interface HTMLTimeElement : HTMLElement {
    [HTMLConstructor] constructor();
    [CFReactions] 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 child text content of the `time` element.

The `datetime` value of a `time` element must match one of the following syntaxes.

**A valid month string**

Example

```html
<time>2011-11</time>
```

**A valid date string**

Example

```html
<time>2011-11-18</time>
```

**A valid yearless date string**

Example

```html
<time>11-18</time>
```

**A valid time string**

Example

```html
<time>14:54</time>
```

Example

```html
<time>14:54:39</time>
```
A valid local date and time string

Example

\(<\text{time}>14:54:39.929</time>\)

Example

\(<\text{time}>2011-11-18T14:54</time>\)

Example

\(<\text{time}>2011-11-18T14:54:39</time>\)

Example

\(<\text{time}>2011-11-18T14:54:39.929</time>\)

Example

\(<\text{time}>2011-11-18 14:54</time>\)

Example

\(<\text{time}>2011-11-18 14:54:39</time>\)

Example

\(<\text{time}>2011-11-18 14:54:39.929</time>\)

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

\(<\text{time}>Z</time>\)

Example

\(<\text{time}>+0000</time>\)

Example

\(<\text{time}>+00:00</time>\)

Example

\(<\text{time}>-0800</time>\)

Example

\(<\text{time}>-08:00</time>\)
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 saving 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]{2277}

A valid global date and time string

Example

<time>2011-11-18T14:54Z</time>

Example

<time>2011-11-18T14:54:39Z</time>

Example

<time>2011-11-18T14:54:39.929Z</time>

Example

<time>2011-11-18T14:54+0000</time>

Example

<time>2011-11-18T14:54:39+0000</time>

Example

<time>2011-11-18T14:54:39.929+0000</time>

Example

<time>2011-11-18T06:54-0800</time>

Example

<time>2011-11-18T06:54-0800</time>

Example

<time>2011-11-18T06:54:39-0800</time>

Example

<time>2011-11-18T06:54:39.929-0800</time>
Example

<time>2011-11-18T06:54:08:00</time>

Example

<time>2011-11-18T06:54:39-08:00</time>

Example

<time>2011-11-18T06:54:39.929-08:00</time>

Example

<time>2011-11-18 14:54Z</time>

Example

<time>2011-11-18 14:54:39Z</time>

Example

<time>2011-11-18 14:54:39.929Z</time>

Example

<time>2011-11-18 14:54+0000</time>

Example

<time>2011-11-18 14:54:39+0000</time>

Example

<time>2011-11-18 14:54:39.929+0000</time>

Example

<time>2011-11-18 14:54+00:00</time>

Example

<time>2011-11-18 14:54:39+00:00</time>

Example

<time>2011-11-18 14:54:39.929+00:00</time>

Example

<time>2011-11-18 06:54-0800</time>

Example

<time>2011-11-18 06:54:39-0800</time>

Example

<time>2011-11-18 06:54:39.929-0800</time>
A valid week string\footnote{p81}

Example

\begin{verbatim}
<time>2011-W47</time>
\end{verbatim}

Four or more ASCII digits, at least one of which is not U+0030 DIGIT ZERO (0)

Example

\begin{verbatim}
<time>2011</time>
\end{verbatim}

Example

\begin{verbatim}
<time>0001</time>
\end{verbatim}

A valid duration string\footnote{p82}

Example

\begin{verbatim}
<time>PT4H18M3S</time>
\end{verbatim}

Example

\begin{verbatim}
<time>4h 18m 3s</time>
\end{verbatim}

The machine-readable equivalent of the element's contents must be obtained from the element's \texttt{datetime value}\footnote{p262} by using the following algorithm:

1. If parsing a month string\footnote{p74} from the element's \texttt{datetime value}\footnote{p262} returns a \texttt{month}\footnote{p74}, that is the machine-readable equivalent; return.

2. If parsing a date string\footnote{p75} from the element's \texttt{datetime value}\footnote{p262} returns a \texttt{date}\footnote{p75}, that is the machine-readable equivalent; return.

3. If parsing a yearless date string\footnote{p76} from the element's \texttt{datetime value}\footnote{p262} returns a \texttt{yearless date}\footnote{p75}, that is the machine-readable equivalent; return.
4. If parsing a time string from the element's datetime value returns a time, that is the machine-readable equivalent; return.

5. If parsing a local date and time string from the element's datetime value returns a local date and time, that is the machine-readable equivalent; return.

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; return.

7. If parsing a global date and time string from the element's datetime value returns a global date and time, that is the machine-readable equivalent; return.

8. If parsing a week string from the element's datetime value returns a week, that is the machine-readable equivalent; return.

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; return.

10. If parsing a duration string from the element's datetime value returns a duration, that is the machine-readable equivalent; return.

11. There is no machine-readable equivalent.

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

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:

```html
div class="vevent">
  a href="http://www.web2con.com/" class="url">
  <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

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.

```html
<article itemscope itemtype="https://n.example.org/rfc4287">
  h1 itemprop="title">Big tasks</h1>
  footer>
  Published <time itemprop="published" datetime="2009-08-29">two days ago</time>.
</footer>
  p itemprop="content">Today, I went out and bought a bike for my kid.</p>
</article>
```

Example

In this example, another article's publication date is marked up using `time`, this time using the schema.org microdata vocabulary:

```html
<article itemscope itemtype="http://schema.org/BlogPosting">
  h1 itemprop="headline">Small tasks</h1>
```
The `<time>` element is used to encode a date in the ISO8601 format, for later processing by a script:

```
Our first date was <time datetime="2006-09-23">a Saturday</time>.
```

In this second snippet, the value includes a time:

```
We stopped talking at <time datetime="2006-09-24T05:00-07:00">5am the next morning</time>.
```

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.

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>.
```

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>.
```

### 4.5.15 The `code` element

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 up 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

The following example shows how the element can be used in a paragraph to mark up element names and computer code, including punctuation.

```html
<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

The following example shows how a block of code could be marked up using the `pre` and `code` elements.

```html
<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.

---

### 4.5.16 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

#### Accessibility considerations:

- For authors
- For implementers

#### 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

In the paragraph below, the letter "n" is being used as a variable in prose:

```html
<p>If there are <var>n</var> pipes leading to the ice cream factory then I expect at least <em>n</em> 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

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.

```html
<figure>
  <math>
    \sqrt{b^2 + c^2} = \sqrt{a^2 + c^2} \text{ Using Pythagoras' theorem to solve for the hypotenuse of a triangle with sides } b \text{ and } c \text{ and } a \text{ of }
  </math>
  <figcaption>
    The teacher looked pleased.
  </figcaption>
</figure>
```

Example

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:

```html
<p>Then she turned to the blackboard and picked up the chalk. After a few moment's thought, she wrote <var>E</var> = <var>m</var> \times <var>c</var>². The teacher looked pleased.</p>
```

4.5.17 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.

270
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**
This example shows the `samp` element being used inline:

```html
<p>The computer said <samp>Too much cheese in tray two</samp> but I didn't know what that meant.</p>
```

**Example**
This second example shows a block of sample output from a console program. 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.

```html
<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 1686 unknown
</samp></pre>
```

**Example**
This third example shows a block of input and its respective output. The example uses both `code` and `samp` elements.

```html
<pre>
<code class="language-javascript">console.log(2.3 + 2.4)</code>
<samp>4.699999999999999</samp>
</pre>
```

### 4.5.18 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
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**

Here the `kbd` element is used to indicate keys to press:

```
<p>To make George eat an apple, press <kbd><kbd><kbd>Shift</kbd></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><samp>Eat Apple...</samp></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>
```
Accessibility considerations:
The `sub` element: for authors; for implementers.
The `sup` element: for authors; for implementers.

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 `sup` and `sub` 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

```
<p>Their names are
  <span lang="fr"><abbr>M</abbr>lle</span> Gwendoline and
  <span lang="fr"><abbr>M</abbr>me</span> Denise.</p>
```

The `sub` element can be used inside a `var` element, for variables that have subscripts.

Example

```
<p>The coordinate of the \(i\)th point is
  \((x_{i}, y_{i})\).
  For example, the 10th point has coordinate
  \((x_{10}, y_{10})\).</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

```
\[ f(x, n) = \log_{4}(x^{n}) \]
```

4.5.20 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

```
Accessibility considerations:
For authors.
For implementers.

DOM interface:
Uses HTML Element.

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**

The examples below show uses of the `i` element:

```html
<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.

```html
<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>Finally one night he picked up the courage to speak with her--</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.

4.5.21 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.
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**

The following example shows a use of the `b` element to highlight key words without marking them up as important:

```html
<p>The <b>frobonitor</b> and <b>barbinator</b> components are fried.</p>
```

**Example**

In the following example, objects in a text adventure are highlighted as being special by use of the `b` element.

```html
<p>You enter a small room. Your <b>sword</b> glows brighter. A <b>rat</b> scurries past the corner wall.</p>
```

**Example**

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:

```html
<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>
</article>
```

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**

The following would be incorrect usage:

```html
<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.
4.5.22 The u element

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.

**Example**

In this example, a *u* element is used to mark a word as misspelt:

```html
<p>The<u>see</u> is full of fish.</p>
```

4.5.23 The mark element

The *mark* element represents a span of text with explicit textual annotations.
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

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:
<blockquote lang="en-GB">
<p>
Look around and you will find, no-one's really <mark>colour</mark> blind.</p>
</blockquote>
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

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:

```
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>.
```

Example

In the following snippet, a paragraph of text refers to a specific part of a code fragment.

```
The highlighted part below is where the error lies:
<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:

```
The highlighted part below is where the error lies:
<pre><code><span class="keyword">var</span> i: Integer;
begin
  <span class="ident">i</span> := <mark>1.1</mark>;
end.</code></pre>
```

Example
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.

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

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.
4.5.24 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.

**Accessibility considerations**:
- For authors
- For implementers

**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**
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").

```html
<ul>
  <li>User <bdi>jcranmer</bdi>: 12 posts.</li>
  <li>User <bdi>hober</bdi>: 5 posts.</li>
  <li>User <bdi>إيان</bdi>: 3 posts.</li>
</ul>
```

- User jcranmer: 12 posts.
- User hober: 5 posts.
- User إيان: 3 posts.

When using the **bdi** element, the username acts as expected.

- User jcranmer: 12 posts.
- User hober: 5 posts.
- User إيان: 3 posts.

If the **bdi** element were to be replaced by a **b** element, the username would confuse the bidirectional algorithm and the third
4.5.25 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.

**Accessibility considerations**: For authors. For implementers.

**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.

4.5.26 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.

**Accessibility considerations**: For authors.
For implementers.

**DOM interface**

```idl
[Exposed=Window]
interface HTMLSpanElement : HTMLElement {
    [HTMLConstructor] constructor();
};
```

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

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:

```html
<pre><code class="lang-c">for</code>(<code class="ident">j</code> = 0; <code class="ident">j</code> &lt; 256; <code class="ident">j</code>++) {
    <code class="ident">i_t3</code> = (<code class="ident">i_t3</code> &amp; 0x1ffff) | (<code class="ident">i_t3</code> &lt;&lt; 17);
    <code class="ident">i_t6</code> = (((((<code class="ident">i_t3</code>) &gt;&gt; 3) ^ <code class="ident">i_t3</code>) &gt;&gt; 1) ^ <code class="ident">i_t3</code>)) &gt;&gt; 8) ^ <code class="ident">i_t3</code>;
    <code class="keyword">if</code>(<code class="ident">i_t6</code> == <code class="ident">i_t1</code>)
        <code class="keyword">break</code>;
} <code class="keyword"></code></code></pre>
```

4.5.27 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

**Accessibility considerations**

- For authors
- For implementers

**DOM interface**

```idl
[Exposed=Window]
interface HTMLBRElement : HTMLElement {
    [HTMLConstructor] constructor();
    // also has obsolete members
};
```

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**
The following example is correct usage of the *br* element:

```html
<p>P. Sherman<br>42 Wallaby Way<br>Sydney</p>
```

*br* elements must not be used for separating thematic groups in a paragraph.

**Example**
The following examples are non-conforming, as they abuse the *br* element:

```html
<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:

```html
<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.

4.5.28 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.
The `wbr` element represents a line break opportunity.

**Example**

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.

```html
<p>So then she 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>
```

Any content inside `wbr` elements must not be considered part of the surrounding text.

**Example**

```javascript
var wbr = document.createElement("wbr");
wbr.textContent = "This is wrong";
document.body.appendChild(wbr);
```

**Note**

This element has rendering requirements involving the bidirectional algorithm.

---

### 4.5.29 Usage summary

This section is non-normative.

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a</code></td>
<td>Hyperlinks</td>
<td>Visit my <code>&lt;a href=&quot;drinks.html&quot;&gt;drinks&lt;/a&gt;</code> page.</td>
</tr>
<tr>
<td><code>em</code></td>
<td>Stress emphasis</td>
<td>I must say I <code>&lt;em&gt;adore&lt;/em&gt;</code> lemonade.</td>
</tr>
</tbody>
</table>
| `strong` | Importance | This tea is `<strong>very hot</strong>`.
| `small` | Side comments | These grapes are made into wine. `<small>Alcohol is addictive</small>`.
| `s` | Inaccurate text | Price: `<s>£4.50</s> £2.00!` |
| `cite` | Titles of works | The case `<cite>Hugo v. Danielle</cite>` is relevant here. |
| `q` | Quotations | The judge said `<q>You can drink water from the fish tank</q>` but advised against it. |
| `dfn` | Defining instance | The term `<dfn>organic food</dfn>` refers to food produced without synthetic chemicals. |
| `abbr` | Abbreviations | Organic food in Ireland is certified by the `<abbr title="Irish Organic Farmers and Growers Association">IOFGA</abbr>`.
| `ruby` | Ruby annotations | `<ruby>01 <rt>Orange Juice</rt></ruby>` |
| `data` | Machine-readable equivalent | Available starting today! `<data value="022014640201">North Coast Organic Apple Cider</data>` |
| `time` | Machine-readable equivalent of date-or time-related data | Available starting on `<time datetime="2011-11-18">November 18th</time>`! |
### 4.6 Links  

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>Computer code</td>
<td>The <code>&lt;code&gt;fruitdb&lt;/code&gt;</code> program can be used for tracking fruit production.</td>
</tr>
<tr>
<td>var</td>
<td>Variables</td>
<td>If there are <code>&lt;var&gt;n&lt;/var&gt;</code> fruit in the bowl, at least `&lt;var&gt;n&lt;/var&gt;+2 will be ripe.</td>
</tr>
<tr>
<td>samp</td>
<td>Computer output</td>
<td>The computer said <code>&lt;samp&gt;Unknown error -3&lt;/samp&gt;</code>.</td>
</tr>
<tr>
<td>kbd</td>
<td>User input</td>
<td>Hit <code>&lt;kbd&gt;Fl&lt;/kbd&gt;</code> to continue.</td>
</tr>
<tr>
<td>sub</td>
<td>Subscripts</td>
<td>Water is H&lt;sub&gt;2&lt;/sub&gt;O.</td>
</tr>
<tr>
<td>sup</td>
<td>Superscripts</td>
<td>The Hydrogen in heavy water is usually <code>&lt;sup&gt;H&lt;sub&gt;2&lt;/sub&gt;&lt;/sup&gt;</code>-O.</td>
</tr>
<tr>
<td>i</td>
<td>Alternative voice</td>
<td>Lemonade consists primarily of <code>&lt;i&gt;Citrus limon&lt;/i&gt;</code>-o.</td>
</tr>
<tr>
<td>b</td>
<td>Keywords</td>
<td>Take a <code>&lt;b&gt;lemon&lt;/b&gt;</code> and squeeze it with a <code>&lt;b&gt;juicer&lt;/b&gt;</code>.</td>
</tr>
<tr>
<td>u</td>
<td>Annotations</td>
<td>The mixture of apple juice and <code>&lt;u class=&quot;spelling&quot;&gt;elderflower&lt;/u&gt;</code> juice is very pleasant.</td>
</tr>
<tr>
<td>mark</td>
<td>Highlight</td>
<td>Elderflower cordial, with one <code>&lt;mark&gt;part&lt;/mark&gt;</code> cordial to ten <code>&lt;mark&gt;part&lt;/mark&gt;</code>s water, stands as <code>&lt;mark&gt;part&lt;/mark&gt;</code> from the rest.</td>
</tr>
<tr>
<td>bdi</td>
<td>Text directionality isolation</td>
<td>The recommended restaurant is <code>&lt;bdi lang=&quot;fr&quot;&gt;My Juice Café (At The Beach)&lt;/bdi&gt;</code>.</td>
</tr>
<tr>
<td>bdo</td>
<td>Text directionality formatting</td>
<td>The proposal is to write English, but in reverse order. &quot;Juice&quot; would become <code>&quot;&lt;bdo dir=&quot;rtl&quot;&gt;Juice&lt;/bdo&gt;&quot;</code>-o.</td>
</tr>
<tr>
<td>span</td>
<td>Other</td>
<td>In French we call it <code>&lt;span lang=&quot;fr&quot;&gt;sirop de sureau&lt;/span&gt;</code>.</td>
</tr>
<tr>
<td>br</td>
<td>Line break</td>
<td>Simply Orange Juice Company&lt;br&gt;Apopka, FL 32703&lt;br&gt;U.S.A.</td>
</tr>
<tr>
<td>wbr</td>
<td>Line breaking opportunity</td>
<td><a href="http://www.simply">www.simply</a>&lt;wbr&gt;orange&lt;wbr&gt;juice.com</td>
</tr>
</tbody>
</table>

#### 4.6.1 Introduction

Links are a conceptual construct, created by `a`, `area`, `form`, 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. All external resource links have a fetch and process the linked resource algorithm which describes how the resource is obtained.

**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.

Similarly, for `form` elements with a `rel` attribute, links must be created for the keywords of the `rel` attribute as defined for those keywords in the link types section. `form` elements that 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 can have one or more `hyperlink annotations` that modify the processing semantics of that hyperlink.

---

284
4.6.2 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 `ping` attribute, if present, gives the URLs of the resources that are interested in being notified if the user follows the hyperlink. The value must be a set of space-separated tokens, each of which must be a valid non-empty URL whose scheme is an HTTP(S) scheme. The value is used by the user agent for hyperlink auditing.

The `rel` attribute on `a` and `area` elements controls what kinds of links the elements create. The attribute's value must be an unordered set of unique 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,noopener, and opener. `rel`'s supported tokens must only include the tokens from this list that the user agent implements the processing model for.

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. 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 string. 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.

The `referrerpolicy` attribute is a referrer policy attribute. Its purpose is to set the referrer policy used when following hyperlinks, [REFERRERPOLICY].

4.6.3 API for `a` and `area` elements

[IDL] interface mixin HTMLHyperlinkElementUtils {
  [CEReactions] stringifier attribute USVString href;
  readonly attribute USVString origin;
  [CEReactions] attribute USVString protocol;
}
For web developers (non-normative)

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 runs these steps:
1. If this element's `href` content attribute is absent, set this element's `url` to null.

2. Otherwise, parse this element's `href` content attribute value relative to this element's `node document` if parsing is successful, set this element's `url` to the result; otherwise, 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`.

An element implementing the `HTMLHyperlinkElementUtils` mixin has an associated `reinitialize url` algorithm, which runs these steps:

1. If element's `url` is non-null, its `scheme` is "blob", and its `cannot-be-a-base-URL` is true, 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. Reinitialize `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. Reinitialize `url`.
2. If this element's `url` is null, return the empty string.
3. Return the serialization of this element's `url`'s `origin`.

The `protocol` attribute's getter must run these steps:

1. Reinitialize `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. Reinitialize `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`.

Note

Because the URL parser ignores multiple consecutive colons, providing a value of "https:" (or even "https:::"") is the same as providing a value of "https:"

4. Update `href`.

The username attribute's getter must run these steps:
1. Reinitialize 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. Reinitialize url.
2. Let url be this element's url.
3. If url is null or url cannot have a username/password/port, then return.
4. Set the username, given url and the given value.
5. Update href.

The password attribute's getter must run these steps:

1. Reinitialize url.
2. Let url be this element's url.
3. If url is null, then return the empty string.
4. Return url's password.

The password attribute's setter must run these steps:

1. Reinitialize url.
2. Let url be this element's url.
3. If url is null or url cannot have a username/password/port, then return.
4. Set the password, given url and the given value.
5. Update href.

The host attribute's getter must run these steps:

1. Reinitialize 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. Reinitialize url.
2. Let url be this element's url.
3. If url is null or url's cannot-be-a-base-URL is true, then return.
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. Reinitialize 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. Reinitialize url.
2. Let url be this element's url.
3. If url is null or url's cannot-be-a-base-URL is true, then return.
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. Reinitialize 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. Reinitialize url.
2. Let url be this element's url.
3. If url is null or url cannot have a username/password/port, then return.
4. If the given value is the empty string, then set url's port to null.
5. Otherwise, basic URL parse the given value, with url as url and port state as state override.
6. Update href.

The pathname attribute's getter must run these steps:

1. Reinitialize url.
2. Let url be this element's url.
3. If url is null, return the empty string.
4. If url's cannot-be-a-base-URL is true, then return url's path[0].
5. If url's path is empty, then return the empty string.
6. 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. Reinitialize url.
2. Let url be this element's url.
3. If url is null or url's cannot-be-a-base-URL is true, then return.
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. Reinitialize 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`\(^{299}\) attribute's setter must run these steps:

1. Reinitialize `url`\(^{287}\).
2. Let `url` be this element's `url`\(^{286}\).
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:
   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 null, this element's `node document`'s `document`'s `character encoding`, `url` as `url`, and `query state` as `state override`.
6. Update `href`\(^{287}\).

The `hash` attribute's getter must run these steps:

1. Reinitialize `url`\(^{287}\).
2. Let `url` be this element's `url`\(^{286}\).
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`\(^{290}\) attribute's setter must run these steps:

1. Reinitialize `url`\(^{287}\).
2. Let `url` be this element's `url`\(^{286}\).
3. If `url` is null, then return.
4. If the given value is the empty string, set `url`'s `fragment` to null.
5. Otherwise:
   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`\(^{287}\).

4.6.4 Following hyperlinks \(^{5\text{p}29}\)

An element `element` cannot navigate if one of the following is true:

- `element`'s `node document` is not fully active\(^{5817}\)
- `element` is not an `a`\(^{239}\)` element and is not connected.

**Note**

This is also used by form submission\(^{5\text{p}24}\) for the `form`\(^{5486}\)` element. The exception for `a`\(^{239}\)` elements is for compatibility with web content.

To get an element's noopenner, given an `a`\(^{239}\), `area`\(^{444}\), or `form`\(^{5486}\)` element `element` and a string `target`, run these steps:
1. If element's link types include the `noopener` or `noreferrer` keyword, then return true.

2. If element's link types do not include the `opener` keyword and target is an ASCII case-insensitive match for "_blank", then return true.

3. Return false.

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. If subject cannot navigate, then return.

2. Let replace be false.

3. Let source be subject's node document's browsing context.

4. Let targetAttributeValue be the empty string.

5. If subject is an `a` or `area` element, then set targetAttributeValue to the result of getting an element's target given subject.

6. Let noopener be the result of getting an element's noopener with subject and targetAttributeValue.

7. Let target and windowType be the result of applying the rules for choosing a browsing context given targetAttributeValue, source, and noopener.

8. If target is null, then return.

9. Parse a URL given subject's href attribute, relative to subject's node document.

   Otherwise, if parsing the URL failed, the user agent may report the error to the user in a user-agent-specific manner, may queue an element task on the DOM manipulation task source given subject to navigate to an error page to report the error, or may ignore the error and do nothing. In any case, the user agent must then return.

11. If there is a hyperlink suffix, append it to URL.

12. Let request be a new request whose url is URL and whose referrer policy is the current state of subject's referrer policy content attribute.

13. If subject's link types includes the `noreferrer` keyword, then set request's referrer to "no-referrer".

14. Let historyHandling be "replace" if windowType is not "existing or none"; otherwise, "default".

15. Queue an element task on the DOM manipulation task source given subject to navigate to request with historyHandling set to historyHandling and the source browsing context set to source.

4.6.5 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.)

The following allowed to download algorithm takes an initiator browsing context and an instantiator browsing context, and returns a boolean indicating whether or not downloading is allowed:
1. If the *initiator browsing context*’s sandboxing flags has the sandboxed downloads browsing context flag set, then return false.

2. If the *instantiator browsing context* is non-null, and its sandboxing flags has the sandboxed downloads browsing context flag set, then return false.

3. Optionally, the user agent may return false, if it believes doing so would safeguard the user from a potentially hostile download.

4. Return true.

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. If *subject* cannot navigate, then return.

2. Run the **allowed to download** algorithm with the *subject’s node document’s browsing context* and null. If the algorithm returns false, then return.

3. **Parse a URL** given *subject’s* href attribute, relative to *subject’s node document*.

4. 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 return.

5. Otherwise, let URL be the resulting URL string.

6. If there is a hyperlink suffix, append it to URL.

7. Run these steps **in parallel**:  
   1. Let request be a new request whose url is URL, client is entry settings object, initiator is “download”, destination is the empty string, and whose synchronous flag and use-URL-credentials flag are set.
   2. Handle the result of fetching request 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. Otherwise, it 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 undefined 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]²¹²⁷

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]²¹²⁷

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 an implementation-defined manner, and jump to the step labeled sanitize below.

12. 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.

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.

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.

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.

18. If claimed type and named type are the same type (i.e. the type given by the resource's Content-Type metadata is consistent with the type given by filename's extension), then return filename as the file name.

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.").

   This last step would make it impossible to download executables, which might not be desirable. As always, implementers 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.
4.6.5.1 Hyperlink auditing

If a hyperlink, created by an a or area element has a ping attribute, and the user follows the hyperlink, and the value of the element's href attribute can be parsed, relative to the element's node document, without failure, then the user agent must take the ping attribute's value, split that string on ASCII whitespace, parse each resulting token relative to the element's node document, and then run these steps for each resulting URL record ping URL, ignoring tokens that fail to parse:

1. If ping URL's scheme is not an HTTP(S) scheme, then return.
2. Optionally, return. (For example, the user agent might wish to ignore any or all ping URLs in accordance with the user's expressed preferences.)
3. Let request be a new request whose url is ping URL, method is `POST`, body is `PING`, client is the environment settings object containing the hyperlink, destination is the empty string, credentials mode is "include", referrer is "no-referrer", and whose use-URL-credentials flag is set.
4. Let target URL be the resulting URL string obtained from parsing the value of the element's href attribute and then:
   → If the URL of the Document object containing the hyperlink being audited and ping URL have the same origin:
     request must include a `Ping-From` header with, as its value, the URL of the document containing the hyperlink, and a `Ping-To` HTTP header with, as its value, the target URL.
   → Otherwise request must include a `Ping-To` HTTP header with, as its value, target URL. request does not include a `Ping-From` header.
5. Fetch request.

This may be done in parallel with the primary fetch, and is independent of the result of that fetch.

User agents should allow the user to adjust this behavior, for example in conjunction with a setting that disables the sending of HTTP `Referer` (sic) headers. Based on the user's preferences, UAs may either ignore the ping attribute altogether, or selectively ignore URLs in the list (e.g. ignoring any third-party URLs); this is explicitly accounted for in the steps above.

User agents must ignore any entity bodies returned in the responses. User agents may close the connection prematurely once they start receiving a response body.

When the ping attribute is present, user agents should clearly indicate to the user that following the hyperlink will also cause secondary requests to be sent in the background, possibly including listing the actual target URLs.

Example

For example, a visual user agent could include the hostnames of the target ping URLs along with the hyperlink's actual URL in a status bar or tooltip.

Note

The ping attribute is redundant with pre-existing technologies like HTTP redirects and JavaScript in allowing web pages to track which off-site links are most popular or allowing advertisers to track click-through rates.

However, the ping attribute provides these advantages to the user over those alternatives:

- It allows the user to see the final target URL unobscured.
- It allows the UA to inform the user about the out-of-band notifications.
- It allows the user to disable the notifications without losing the underlying link functionality.
The following table summarizes the link types that are defined by this specification, by their corresponding keywords. 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, an area, or form element, the element’s rel attribute must be split on ASCII whitespace. The resulting tokens are the keywords for the link types that apply to that element.

Except where otherwise specified, a keyword must not be specified more than once per rel attribute.

Some of the sections that follow the table below list synonyms for certain keywords. The indicated synonyms are to be handled as specified by user agents, but must not be used in documents (for example, the keyword "copyright").

Keywords are always ASCII case-insensitive, and must be compared as such.

### Example

Thus, rel="next" is the same as rel="NEXT".

Keywords that are body-ok affect whether link elements are allowed in the body. The body-ok keywords are dns-prefetch, modulepreload, pingback, preconnect, prefetch, preload, prerender, and stylesheet.

New link types that are to be implemented by web browsers are to be added to this standard. The remainder can be registered as extensions.

<table>
<thead>
<tr>
<th>Link type</th>
<th>Effect on...</th>
<th>body-ok</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alternate</td>
<td>Hyperlink</td>
<td>not allowed</td>
<td>Gives alternate representations of the current document.</td>
</tr>
<tr>
<td>canonical</td>
<td>Hyperlink</td>
<td>not allowed</td>
<td>Gives the preferred URL for the current document.</td>
</tr>
<tr>
<td>author</td>
<td>Hyperlink</td>
<td>not allowed</td>
<td>Gives a link to the author of the current document or article.</td>
</tr>
<tr>
<td>bookmark</td>
<td>Hyperlink</td>
<td>not allowed</td>
<td>Gives the permalink for the nearest ancestor section.</td>
</tr>
<tr>
<td>dns-prefetch</td>
<td>ExternalResource</td>
<td>not allowed</td>
<td>Specifies that the user agent should preemptively perform DNS resolution for the target resource’s origin.</td>
</tr>
<tr>
<td>external</td>
<td>not allowed</td>
<td>Annotation</td>
<td>Indicates that the referenced document is not part of the same site as the current document.</td>
</tr>
<tr>
<td>help</td>
<td>Hyperlink</td>
<td></td>
<td>Provides a link to context-sensitive help.</td>
</tr>
<tr>
<td>icon</td>
<td>ExternalResource</td>
<td>not allowed</td>
<td>Imports an icon to represent the current document.</td>
</tr>
<tr>
<td>manifest</td>
<td>ExternalResource</td>
<td>not allowed</td>
<td>Imports or links to an application manifest. [MANIFEST]</td>
</tr>
<tr>
<td>modulepreload</td>
<td>ExternalResource</td>
<td>not allowed</td>
<td>Specifies that the user agent must preemptively fetch the module script and store it in the document’s module map for later evaluation. Optionally, the module’s dependencies can be fetched as well.</td>
</tr>
<tr>
<td>license</td>
<td>Hyperlink</td>
<td></td>
<td>Indicates that the main content of the current document is covered by the copyright license described by the referenced document.</td>
</tr>
<tr>
<td>next</td>
<td>Hyperlink</td>
<td></td>
<td>Indicates that the current document is a part of a series, and that the next document in the series is the referenced document.</td>
</tr>
<tr>
<td>nofollow</td>
<td>not allowed</td>
<td>Annotation</td>
<td>Indicates that the current document’s original author or publisher does not endorse the referenced document.</td>
</tr>
</tbody>
</table>
### 4.6.6.1 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.

Example

Here, a set of `link` elements provide some style sheets:

```html
<!-- 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">
```

- **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).

For the purposes of feed autodiscovery, user agents should consider all `link` elements in the document with the
The **alternate** keyword used and with their **type** attribute set to the value `application/rss+xml` or the value `application/atom+xml`. If the user agent has the concept of a default syndication feed, the first such element (in tree order) should be used as the default.

**Example**

The following **link** elements give syndication feeds for a blog:

```html
<link rel="alternate" type="application/atom+xml" href="posts.xml" title="Cool Stuff Blog">
<link rel="alternate" type="application/atom+xml" href="posts.xml?category=robots" title="Cool Stuff Blog: robots category">
<link rel="alternate" type="application/atom+xml" href="comments.xml" title="Cool Stuff Blog: Comments">
```

Such **link** elements would be used by user agents engaged in feed autodiscovery, with the first being the default (where applicable).

The following example offers various different syndication feeds to the user, using **a** elements:

```html
<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>
```

These links would not be used in feed autodiscovery.

**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 document element's **language** attribute, 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**

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:

```html
<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">
```

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.
### 4.6.6.2 Link type "author" §p298

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.

**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.

### 4.6.6.3 Link type "bookmark" §p298

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**

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.

```html
...<body>
  <h1>Example of permalinks</h1>
  <div id="a">
    <h2>First example</h2>
    <p><a href="a.html" rel="bookmark">This permalink applies to 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>
  <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>
</body>...
```

### 4.6.6.4 Link type "canonical" §p298

The canonical keyword may be used with link element. This keyword creates a hyperlink.

The canonical keyword indicates that URL given by the href attribute is the preferred URL for the current document. That helps search engines reduce duplicate content, as described in more detail in The Canonical Link Relation. [RFC6596]
The `dns-prefetch` keyword may be used with `link` elements. This keyword creates an `external resource link`. This keyword is body-ok.

The `dns-prefetch` keyword indicates that preemptively performing DNS resolution for the origin of the specified resource is likely to be beneficial, as it is highly likely that the user will require resources located at that origin, and the user experience would be improved by preempting the latency costs associated with DNS resolution. User agents must implement the processing model of the `dns-prefetch` keyword described in Resource Hints. [RESOURCEHINTS]

There is no default type for resources given by the `dns-prefetch` keyword.

The `external` keyword may be used with `a`, `area`, and `form` 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 `external` keyword indicates that the link is leading to a document that is not part of the site that the current document forms a part of.

The `help` keyword may be used with `link`, `a`, `area`, and `form` elements. This keyword creates a hyperlink.

For `a`, `area`, and `form` 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

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.

```html
<p><label> Topic: <input name=topic> <a href=help/topic.html rel=help>(Help)</a></label></p>
```

For `a` 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.

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 \texttt{sizes} keywords represent icon sizes in raw pixels (as opposed to \texttt{CSS pixels}).

\textbf{Note}

An icon that is 50 \texttt{CSS pixels} wide intended for displays with a device pixel density of two device pixels per \texttt{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 \textit{split the attribute's value on ASCII whitespace}, and must then parse each resulting keyword to determine what it represents.

The \texttt{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. Return for that keyword.
- Let \textit{width string} be the string before the "\texttt{x}" or "\texttt{X}".
- Let \textit{height string} be the string after the "\texttt{x}" or "\texttt{X}".
- If either \textit{width string} or \textit{height string} start with a U+0030 DIGIT ZERO (0) character or contain any characters other than \texttt{ASCII digits}, then this keyword doesn't represent anything. Return for that keyword.
- Apply the rules for parsing non-negative integers\textsuperscript{66} to \textit{width string} to obtain \textit{width}.
- Apply the rules for parsing non-negative integers\textsuperscript{66} to \textit{height string} to obtain \textit{height}.
- The keyword represents that the resource contains a bitmap icon with a width of \textit{width} device pixels and a height of \textit{height} device pixels.

The keywords specified on the \texttt{sizes} attribute must not represent icon sizes that are not actually available in the linked resource.

The \textit{linked resource fetch setup steps}\textsuperscript{64} for this type of linked resource, given a \texttt{link} element \texttt{el} and \texttt{request} \texttt{request}, are:

1. Set \texttt{request}'s \texttt{destination} to "image".
2. Return true.

In the absence of a \texttt{link} with the \texttt{icon} keyword, for \texttt{Document} objects whose \texttt{URL}'s \texttt{scheme} is an \texttt{HTTP(S) scheme}, user agents may instead run these steps \textit{in parallel}\textsuperscript{42}:

1. Let \texttt{request} be a new \texttt{request} whose \texttt{url} is the \texttt{URL record} obtained by resolving the \texttt{URL} "/favicon.ico" against the \texttt{Document} object's \texttt{URL}, \texttt{client} is the \texttt{Document} object's \texttt{relevant settings object}\textsuperscript{90}, \texttt{destination} is "image", \texttt{synchronous flag} is set, \texttt{credentials mode} is "include", and whose \texttt{use-URL-credentials flag} is set.
2. Let \texttt{response} be the result of fetching \texttt{request}.
3. Use \texttt{response}'s \texttt{unsafe response} as an icon as if it had been declared using the \texttt{icon} keyword.

\textbf{Example}

The following snippet shows the top part of an application with several icons.

\begin{verbatim}
<!DOCTYPE HTML>
<html lang="en">
<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>
</head>
</html>
\end{verbatim}
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 ASCII whitespace).

4.6.6.9 Link type "license" §

The `license` keyword may be used with `link`, `a`, `area`, and `form` 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 does not specify how to distinguish between the main content of a document and content that is not deemed to be part of that main content. The distinction should be made clear to the user.

**Example**

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:

```html
<!DOCTYPE HTML>
<html lang="en">
<head>
<title>Example Pictures: Kissat</title>
<link rel="stylesheet" href="/style/default">
</head>
<body>
<h1>Kissat</h1>
<nav>
<a href="../">Return to photo index</a>
</nav>
<figure>
<img src="/pix/39627052_fd8dcd98b5.jpg">
<figcaption>Kissat</figcaption>
</figure>
<p>One of them has six toes!</p>
<p><small>&lt; a rel="license" href="http://www.opensource.org/licenses/mit-license.php">MIT Licensed</small>&lt; /a></p>
<footer>
<a href="/">Home</a> | <a href="../">Photo index</a>
<p>&copy; copyright 2009 Exampl Pictures. All Rights Reserved.</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 could be made clearer in the styling (e.g. making the license link prominently positioned near the photograph, while having the page copyright in light small text at the foot of the page).

**Synonyms:** For historical reasons, user agents must also treat the keyword "copyright" like the `license` keyword.
4.6.6.10 Link type "manifest"§p30

The `manifest` keyword may be used with `link` elements. This keyword creates an `external resource link`. The `manifest` keyword indicates the manifest file that provides metadata associated with the current document.

There is no default type for resources given by the `manifest` keyword.

The appropriate time to fetch and process the linked resource for this link type is when the user agent deems it necessary. For example, when the user chooses to install the web application. In that case, only the first `link` element in tree order whose `rel` attribute contains the token `manifest` may be used.

A user agent must not delay the load event for this link type.

The linked resource fetch setup steps for this type of linked resource, given a `link` element `el` and `request` `request`, are:

1. Let `context` be `el`'s `node document`'s `browsing context`.
2. If `context` is null, then return false.
3. If `context` is not a `top-level browsing context`, then return false.
4. Set `request`'s `initiator` to "manifest".
5. Set `request`'s `destination` to "manifest".
6. Set `request`'s `mode` to "cors".
7. Set `request`'s `credentials mode` to the CORS settings attribute `credentials mode` for `el`'s `crossorigin` content attribute.
8. Return true.

To process this type of linked resource given a `link` element `el`, boolean `success`, and `response` `response`:

1. If `response`'s `Content-Type metadata` is not a JSON MIME type, then set `success` to false.
2. If `success` is true, then process the manifest given `el` and `response`. [MANIFEST]p1274

4.6.6.11 Link type "modulepreload"§p30

The `modulepreload` keyword may be used with `link` elements. This keyword creates an `external resource link`. This keyword is `body-ok`.

The `modulepreload` keyword is a specialized alternative to the `preload` keyword, with a processing model geared toward preloading `module scripts`. In particular, it uses the specific fetch behavior for module scripts (including, e.g., a different interpretation of the `crossorigin` attribute), and places the result into the appropriate `module map` for later evaluation. In contrast, a similar `external resource link` using the `preload` keyword would place the result in the preload cache, without affecting the document's `module map`.

Additionally, implementations can take advantage of the fact that `module scripts` declare their dependencies in order to fetch the specified module's dependency as well. This is intended as an optimization opportunity, since the user agent knows that, in all likelihood, those dependencies will also be needed later. It will not generally be observable without using technology such as service workers, or monitoring on the server side. Notably, the appropriate `load` or `error` events will occur after the specified module is fetched, and will not wait for any dependencies.

The appropriate times to fetch and process the linked resource for such a link are:

- When the `external resource link` is created on a `link` element that is already `browsing-context connected`.
- When the `external resource link`'s `link` element becomes `browsing-context connected`.
- When the `href` attribute of the `link` element of an `external resource link` that is already `browsing-context connected` is changed.
Unlike some other link relations, changing the relevant attributes (such as `as`, `crossorigin`, and `referrerpolicy`) of such a `link` does not trigger a new fetch. This is because the document's module map has already been populated by a previous fetch, and so re-fetching would be pointless.

The fetch and process the linked resource algorithm for `modulepreload` links, given a `link` element `el`, is as follows:

1. If `el`'s `href` attribute's value is the empty string, then return.
2. Let `destination` be the current state of the `as` attribute (a `destination`), or "script" if it is in no state.
3. If `destination` is not script-like, then queue an element task on the networking task source given the `link` element to fire an event named `error` at the `link` element, and return.
4. Parse a URL given `el`'s `href` attribute, relative to the element's `node document`. If that fails, then return. Otherwise, let `url` be the resulting URL record.
5. Let `settings object` be the `link` element's `node document`'s relevant settings object.
6. Let `credentials mode` be the CORS settings attribute `credentials mode` for the `crossorigin` attribute.
7. Let `cryptographic nonce` be the current value of the element's `[[CryptographicNonce]]` internal slot.
8. Let `integrity metadata` be the value of the `integrity` attribute, if it is specified, or the empty string otherwise.
9. Let `referrer policy` be the current state of the element's `referrerpolicy` attribute.
10. Let `options` be a script fetch options whose `cryptographic nonce` is `cryptographic nonce`, `integrity metadata` is `integrity metadata`, `parser metadata` is "not-parser-inserted", `credentials mode` is `credentials mode`, and `referrer policy` is `referrer policy`.
11. Fetch a modulepreload module script graph given `url`, `destination`, `settings object`, and `options`. Wait until the algorithm asynchronously completes with `result`.
12. If `result` is null, then fire an event named `error` at the `link` element, and return.
13. Fire an event named `load` at the `link` element.

Example

The following snippet shows the top part of an application with several modules preloaded:

```html
<!DOCTYPE html>
<html lang="en">
<title>IRCFog</title>

<link rel="modulepreload" href="app.mjs">
<link rel="modulepreload" href="helpers.mjs">
<link rel="modulepreload" href="irc.mjs">
<link rel="modulepreload" href="fog-machine.mjs">

<script type="module" src="app.mjs">
...

Assume that the module graph for the application is as follows:

```
/\ app.mjs
|   irc.mjs
down  fog-machine.js
up    helpers.mjs
```
Here we see the application developer has used `modulepreload` to declare all of the modules in their module graph, ensuring that the user agent initiates fetches for them all. Without such preloading, the user agent might need to go through multiple network roundtrips before discovering `helpers.mjs`, if technologies such as HTTP/2 Server Push are not in play. In this way, `modulepreload` elements can be used as a sort of “manifest” of the application’s modules.

Example

The following code shows how `modulepreload` links can be used in conjunction with `import()` to ensure network fetching is done ahead of time, so that when `import()` is called, the module is already ready (but not evaluated) in the module map:

```html
<link rel="modulepreload" href="awesome-viewer.mjs">
<button onclick="import('./awesome-viewer.mjs').then(m => m.view())">View awesome thing</button>
```

4.6.6.12 Link type “nofollow”

The `nofollow` keyword may be used with `a`, `area`, and `form` 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.

4.6.6.13 Link type “noopener”

The `noopener` keyword may be used with `a`, `area`, and `form` 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 `noopener` keyword indicates that any newly created `top-level browsing context` which results from following the hyperlink will not be an `auxiliary browsing context`. E.g., its `window.opener` attribute will be null.

Note

See also the processing model where the branching between an `auxiliary browsing context` and a `top-level browsing context` is defined.

Example

This typically creates an `auxiliary browsing context` (assuming there is no existing `browsing context` whose `browsing context name` is "example"):

```html
<a href="help.html" target="example">Help!</a>
```

This creates a `top-level browsing context` that is not an `auxiliary browsing context` (assuming the same thing):

```html
<a href="help.html" target="example" rel="noopener">Help!</a>
```

These are equivalent and only navigate the `parent browsing context`:

```html
<a href="index.html" target="_parent">Home</a>
<a href="index.html" target="_parent" rel="noopener">Home</a>
```
4.6.6.14 Link type "noreferrer" 

The noreferrer keyword may be used with a, area, and form 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 and also implies thenoopener keyword behavior under the same conditions.

See also the processing model where referrer is directly manipulated.

Example

```html
<a href="..." rel="noreferrer" target="_blank"> has the same behavior as <a href="..." rel="noreferrer noopener" target="_blank">.
```

4.6.6.15 Link type "opener"

The opener keyword may be used with a, area, and form 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 keyword indicates that any newly created top-level browsing context which results from following the hyperlink will be an auxiliary browsing context.

See also the processing model.

Example

```html
In the following example the opener is used to allow the help page popup to navigate its opener, e.g., in case what the user is looking for can be found elsewhere. An alternative might be to use a named target, rather than _blank, but this has the potential to clash with existing names.

```html
<a href="..." rel=opener target=_blank>Help!</a>
```

4.6.6.16 Link type "pingback"

The pingback keyword may be used with link elements. This keyword creates an external resource link. This keyword is body-ok.

For the semantics of the pingback keyword, see Pingback 1.0. [PINGBACK]

4.6.6.17 Link type "preconnect"

The preconnect keyword may be used with link elements. This keyword creates an external resource link. This keyword is body-ok.

The preconnect keyword indicates that preemptively initiating a connection to the origin of the specified resource is likely to be beneficial, as it is highly likely that the user will require resources located at that origin, and the user experience would be improved by preempting the latency costs associated with establishing the connection. User agents must implement the processing model of the preconnect keyword described in Resource Hints. [RESOURCEHINTS]

There is no default type for resources given by the preconnect keyword.
4.6.6.18 Link type "prefetch" §p30

The `prefetch` keyword may be used with `link` elements. This keyword creates an external resource link. This keyword is body-ok.

The `prefetch` keyword indicates that preemptively fetching and caching the specified resource is likely to be beneficial, as it is highly likely that the user will require this resource for future navigations. User agents must implement the processing model of the `prefetch` keyword described in Resource Hints. [RESOURCEHINTS]

There is no default type for resources given by the `prefetch` keyword.

4.6.6.19 Link type "preload" §p30

The `preload` keyword may be used with `link` elements. This keyword creates an external resource link. This keyword is body-ok.

The `preload` keyword indicates that the user agent must preemptively fetch and cache the specified resource according to the potential destination given by the `as` attribute (and the priority associated with the corresponding destination), as it is highly likely that the user will require this resource for the current navigation. User agents must implement the processing model of the `preload` keyword described in Preload, as well as in this specification's fetch and process the linked resource algorithm. [PRELOAD]

There is no default type for resources given by the `preload` keyword.

The linked resource fetch setup steps for this type of linked resource, given a `link` element `el` and `request request`, are:

1. Let `as` be the current state of `el`'s `as` attribute.
2. If `as` does not represent a state, return false.
3. Set `request`'s `destination` to the result of translating `as`.
4. If `as` is "image", then:
   1. Let `selected source` and `selected pixel density` be the URL and pixel density that results from selecting an image source given `el`, respectively.
   2. If `selected source` is null, then return false.
   3. Parse `selected source`, relative to `el`'s `node document`. If that fails, then return false. Otherwise, let `url` be the resulting URL record.
   4. Set `request`'s `url` to `url`.
5. Return true.

4.6.6.20 Link type "prerender" §p30

The `prerender` keyword may be used with `link` elements. This keyword creates an external resource link. This keyword is body-ok.

The `prerender` keyword indicates that the specified resource might be required by the next navigation, and so it may be beneficial to not only preemptively fetch the resource, but also to process it, e.g. by fetching its subresources or performing some rendering. User agents must implement the processing model of the `prerender` keyword described in Resource Hints. [RESOURCEHINTS]

There is no default type for resources given by the `prerender` keyword.

4.6.6.21 Link type "search" §p30

The `search` keyword may be used with `link`, `a`, `area`, and `form` elements. This keyword creates a hyperlink.

The `search` keyword indicates that the referenced document provides an interface specifically for searching the document and its...
The stylesheet keyword may be used with link elements. This keyword creates an external resource link that contributes to the styling processing model. This keyword is body-ok.

The specified resource is a CSS style sheet that describes how to present the document.

If the alternate keyword is also specified on the link element, then the link is an alternative style sheet; 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 fetch and process this type of link are:

- When the external resource link is created on a link element that is already browsing-context connected.
- When the external resource link's link element becomes browsing-context connected.
- When the href attribute of the link element of an external resource link that is already browsing-context connected is changed.
- When the disabled attribute of the link element of an external resource link that is already browsing-context connected is set, changed, or removed.
- When the corsorigin attribute of the link element of an external resource link that is already browsing-context connected is set, changed, or removed.
- When the type attribute of the link element of an external resource link that is already browsing-context connected 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 browsing-context connected, but was previously not obtained due to the type attribute specifying an unsupported type, is set, removed, or changed.
- When the external resource link that is already browsing-context connected changes from being an alternative style sheet 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.

The linked resource fetch setup steps for this type of linked resource, given a link element el (ignoring the request) are:

1. If el's disabled attribute is set, then return false.
2. If el contributes a script-blocking style sheet, increment el's node document's script-blocking style sheet counter by 1.
3. Return true.

See issue #968 for plans to use the CSSOM fetch a CSS style sheet algorithm instead of the default fetch and process the linked resource algorithm.

To process this type of linked resource, given a link element el, boolean success, and response response, the user agent must run these steps:

1. If the resource's Content-Type metadata is not text/css, then set success to false.
2. If `el` no longer creates an external resource link that contributes to the styling processing model, or if, since the resource in question was fetched, it has become appropriate to fetch it again, then return.

3. If `el` has an associated CSS style sheet, remove the CSS style sheet.

4. If success is true, then:
   
   1. Create a CSS style sheet with the following properties:
      
      **type**
      
      `text/css`

      **location**
      
      The resulting URL string determined during the fetch and process the linked resource 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. CSSOM defines what happens when the attribute is dynamically set, changed, or removed.

      **title**
      
      The `title` attribute of `element`, if `element` is in a document tree, or the empty string otherwise.

      **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 style sheet and `element`'s explicitly enabled is false; 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.

      This doesn't seem right. Presumably we should be using the response body? Tracked as issue #2997.

The CSS environment encoding is the result of running the following steps: [CSSSYNTAX]

1. If the element has a `charset` attribute, get an encoding from that attribute's value. If that succeeds, return the resulting encoding. [ENCODING]

2. Otherwise, return the document's character encoding. [DOM]

2. Fire an event named `load` at `el`. 

5. Otherwise, fire an event named `error` at `el`.

6. If `el` contributes a script-blocking style sheet, then:
   1. Assert: `el`'s node document's script-blocking style sheet counter is greater than 0.

### 4.6.6.23 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**

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:

```html
<!DOCTYPE HTML>
<html lang="en">
<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**

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).

```html
<!DOCTYPE HTML>
<html lang="en">
<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>
      <img src="https://upload.wikimedia.org/wikipedia/commons/b/b0/
```
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.

### 4.6.6.24 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.

#### 4.6.6.24.1 Link type "next"

The next keyword may be used with link, a, area, and form 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.

When the next keyword is used with a link element, user agents should implement one of the processing models described in Resource Hints, i.e. should process such links as if they were using one of the dns-prefetch, preconnect, prefetch, or prerender keywords. Which resource hint the user agent wishes to use is implementation-dependent; for example, a user agent may wish to use the less-costly preconnect hint when trying to conserve data, battery power, or processing power, or may wish to pick a resource hint depending on heuristic analysis of past user behavior in similar scenarios.

#### 4.6.6.24.2 Link type "prev"

The prev keyword may be used with link, a, area, and form 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.

#### 4.6.6.25 Other link types

Extensions to the predefined set of link types may be registered in the microformats wiki existing-rel-values page.

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.

Effect on... form

One of the following:

**Not allowed**
- The keyword must not be specified on form elements.

**Hyperlink**
- The keyword may be specified on form elements; it creates a hyperlink.

**External Resource**
- The keyword may be specified on form elements; it creates an external resource link.

**Hyperlink Annotation**
- The keyword may be specified on form 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 must 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 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).

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]

4.7 Edits

The ins and del elements represent edits to the document.

4.7.1 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

Accessibility considerations:
For authors.
For implementers.

DOM interface:
Uses HTMLModElement.

The ins element represents an addition to the document.

Example
The following represents the addition of a single paragraph:

<aside>
<ins>
<p> I like fruit. </p>
</ins>

<aside>
<ins>
  <p> Apples are <em>tasty</em>. 
</p>
  <p> So are pears. 
</p>
</ins>
</aside>

As does the following, because everything in the <code>aside</code> element here counts as <code>phrasing content</code> and therefore there is just one <code>paragraph</code>.

<aside>
<ins>
  <p> Apples are <em>tasty</em>. 
</p>
  <p> So are pears. 
</p>
</ins>
</aside>

<ins> elements should not cross implied paragraph boundaries.

**Example**

The following example represents the addition of two paragraphs, the second of which was inserted in two parts. The first <code>ins</code> 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>
  <p> Apples are <em>tasty</em>. 
</p>
</ins>
<ins datetime="2007-12-19 00:00Z">
  <p> So are pears. 
</p>
</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">
  <p> Apples are <em>tasty</em>. 
</p>
</ins>
<ins datetime="2007-12-19 00:00Z">
  <p> So are pears. 
</p>
</ins>
</aside>

**4.7.2 The** <code>del</code> **element**

Categories:
- <code>Flow content</code>
- <code>Phrasing content</code>

Contexts in which this element can be used:
Where <code>phrasing content</code> 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

Accessibility considerations:

For authors.
For implementers.

DOM interface:

Uses HTMLModElement.

The del element represents a removal from the document.

del elements should not cross implied paragraph boundaries.

Example

The following shows a "to do" list where items that have been done are crossed-off with the date and time of their completion.

```html
<h1>To Do</h1>
<ul>
  <li>Empty the dishwasher</li>
  <li>Watch Walter Lewin's lectures</li>
  <li>Download more tracks</li>
  <li>Buy a printer</li>
</ul>
```

4.7.3 Attributes common to ins and del elements

The cite attribute may be used to specify the URL 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 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 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 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:

```
IDL [Exposed=Window]
interface HTMLModElement {
```
The `cite` IDL attribute must reflect the element's `cite` content attribute. The `dateTime` IDL attribute must reflect the element's `datetime` content attribute.

### 4.7.4 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:

```html
<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):

```html
<section>
  This is the first paragraph. <ins>This sentence was inserted.</ins>
  <p>This second paragraph was inserted.</p>
  This sentence was inserted too.<ins> This is the third paragraph in this example.</ins>
  <!-- (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:

```html
<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&gt; 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.
4.7.5 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

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.

```html
<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">Bug 228: Water buffer overflows in April</del></li>
  <li><ins datetime="2008-02-14T12:02Z">Bug 230: Water heater doesn’t use renewable fuels</ins></li>
</ol>
```

Example

In the following example, a list that started with just fruit was replaced by a list with just colors.

```html
<h1>List of fruits</h1>
<ul>
  <li><del>Lime</del><ins>Green</ins></li>
  <li><del>Apple</del></li>
  <li><del>Pear</del></li>
  <li><ins>Teal</ins></li>
  <li><del>Lemon</del><ins>Yellow</ins></li>
  <li><del>Olive</del></li>
  <li><ins>Purple</ins></li>
</ul>
```

4.7.6 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

Here, a table’s row has been added:

```html
<table>
```

This section is non-normative.
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).

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><th>Game publisher</th><th>Verdict</th></tr>
</thead>
<tbody>
<tr><td>Diablo 2</td><td>Blizzard</td><td>8/10</td></tr>
<tr><td>Portal</td><td>Valve</td><td>10/10</td></tr>
<tr><td><ins>Portal 2</ins></td><td><ins>Valve</ins></td><td><ins>10/10</ins></td></tr>
</tbody>
</table>
```

4.8 Embedded content

4.8.1 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 `img` element, optionally intermixed with script-supporting elements.

Tag omission in text/html:
Neither tag is omissible.

Content attributes:
- Global attributes

Accessibility considerations:
- For authors.
- For implementers.

DOM interface:

```idl
defined=Window]
interface HTMLPictureElement : HTMLElement {
[HTMLConstructor] constructor();
};
```

The `picture` element is a container which provides multiple sources to its contained `img` 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. Also, the `picture` element itself does not display anything; it merely provides a context for its contained `img` element that enables it to choose from multiple URLs.

### 4.8.2 The `source` element

**Categories:** None.

**Contexts in which this element can be used:**
- As a child of a `picture` element, before the `img` element.
- 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
  - `srcset` — Images to use in different situations, e.g., high-resolution displays, small monitors, etc.
  - `sizes` — Image sizes for different page layouts
  - `media` — Applicable media

**Accessibility considerations:**
- For authors.
- For implementers.

**DOM interface:**

```idl
[Exposed=Window]
interface HTMLSourceElement : HTMLElement {
    [HTMLConstructor] constructor();

    [CEReactions] attribute USVString src;
    [CEReactions] attribute DOMString type;
    [CEReactions] attribute USVString srcset;
    [CEReactions] attribute DOMString sizes;
    [CEReactions] attribute DOMString media;
};
```

The `source` element allows authors to specify multiple alternative `source` elements or multiple alternative `media resources` for `media elements`. It does not represent anything on its own.

The `type` attribute may be present. If present, the value must be a valid MIME type string.

The remainder of the requirements depend on whether the parent is a `picture` element or a `media element`.

- **source element's parent is a picture element**
  - The `srcset` attribute must be present, and is a `srcset attribute`.

  The `srcset` attribute contributes the `image sources` to the `source set`, if the `source` element is selected.
If the `srcset` attribute has any image candidate strings using a width descriptor, the `sizes` attribute must also be present, and is a `sizes` attribute. The `sizes` attribute contributes the `source size` to the `source set`, if the `source` element is selected.

The `media` attributes may also be present. If present, the value must contain a valid media query list. The user agent will skip to the next `source` element if the value does not match the environment.

The `type` attribute 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 `img` 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 ASCII 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.

* source element's parent is a media element*

The `src` attribute gives the URL 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` 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` attribute gives the type of the `media resource`, to help the user agent determine if it can play this `media resource` before fetching it. The `codecs` parameter, which certain MIME types define, might be necessary to specify exactly how the resource is encoded. [RFC6381]

Example

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"'>
```
The `srcset`\[^{p318}\], `sizes`\[^{p319}\], and `media`\[^{p319}\] attributes must not be present.

If a `source`\[^{p318}\] element is inserted\[^{p344}\] as a child of a `media`\[^{p388}\] element that has no `src`\[^{p390}\] attribute and whose `networkState`\[^{p392}\] has the value `NETWORK_EMPTY`\[^{p392}\], the user agent must invoke the `media`\[^{p388}\] element's `resource selection algorithm`\[^{p394}\].

The IDL attributes `src`, `type`, `srcset`, `sizes` and `media` must reflect\[^{p95}\] the respective content attributes of the same name.

**Example**

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`\[^{p326}\] event on the last `source`\[^{p318}\] element and trigger fallback behavior:

```html
<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>
```
4.8.3 The *img* element

**Categories**:  
Flow content\(^{129}\)  
Phrasing content\(^ {133}\)  
Embedded content\(^ {133}\)  
Form-associated element\(^ {134}\)  
Interactive content\(^ {133}\)  
Palpable content\(^ {134}\)

**Contexts in which this element can be used**\(^ {129}\):  
Where embedded content\(^ {133}\) is expected.

**Content model**\(^ {129}\):  
Nothing\(^ {131}\)

**Tag omission in text/html**\(^ {130}\):  
No *end tag*\(^ {1062}\)

**Content attributes**\(^ {130}\):  
Global attributes\(^ {137}\)  
alt\(^ {322}\) — Replacement text for use when images are not available  
src\(^ {322}\) — Address of the resource  
srcset\(^ {322}\) — Images to use in different situations, e.g., high-resolution displays, small monitors, etc.  
sizes\(^ {322}\) — Image sizes for different page layouts  
crossorigin\(^ {322}\) — How the element handles cross-origin requests  
usemap\(^ {446}\) — Name of *image map*\(^ {446}\) to use  
ismap\(^ {324}\) — Whether the image is a server-side image map  
width\(^ {450}\) — Horizontal dimension  
height\(^ {450}\) — Vertical dimension  
referrerPolicy\(^ {322}\) — Referrer policy for fetches initiated by the element  
decoding\(^ {322}\) — Decoding hint to use when processing this image for presentation  
loading\(^ {322}\) — Used when determining loading deferral

**Accessibility considerations**\(^ {130}\):  
If the element has a non-empty alt\(^ {322}\) attribute: for authors; for implementers.  
Otherwise: for authors; for implementers.

**DOM interface**\(^ {130}\):  

```
IDL  [Exposed=Window,  
LegacyFactoryFunction=Image(optional unsigned long width, optional unsigned long height)]
interface HTMLImageElement : HTMLElement {
  [HTMLConstructor] constructor();

  [CEReactions] attribute DOMString alt;
  [CEReactions] attribute USVString src;
  [CEReactions] attribute USVString srcset;
  [CEReactions] attribute DOMString sizes;
  [CEReactions] attribute DOMString? crossOrigin;
  [CEReactions] attribute DOMString useMap;
  [CEReactions] attribute boolean isMap;
  [CEReactions] attribute unsigned long width;
  [CEReactions] attribute unsigned long height;
  readonly attribute unsigned long naturalWidth;
  readonly attribute unsigned long naturalHeight;
  readonly attribute boolean complete;
  readonly attribute USVString currentSrc;
  [CEReactions] attribute DOMString referrerPolicy;
  [CEReactions] attribute DOMString decoding;
  [CEReactions] attribute DOMString loading;

Promise<undefined> decode();
```

✔ MDN

321
An `<img>` element represents an image.

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 provides equivalent content for those who cannot process images or who have image loading disabled (i.e. it is the `<img>` element's fallback content).

The requirements on the `alt` attribute's value are described in a separate 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, and is a `srcset` attribute.

If the `srcset` attribute is present and has any image candidate strings using a `width descriptor`, the `sizes` attribute must also be present, and is a `sizes` attribute.

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.

The `referrerpolicy` attribute is a referrer policy attribute. Its purpose is to set the referrer policy used when fetching the image.

The decoding attribute indicates the preferred method to decode this image. The attribute, if present, must be an image decoding hint. This attribute's missing value default and invalid value default are both the auto state.

The loading attribute is a lazy loading attribute. Its purpose is to indicate the policy for loading images that are outside the viewport.

When the `loading` attribute's state is changed to the Eager state, the user agent must run these steps:

1. Let `resumptionSteps` be the `<img>` element's lazy load resumption steps.
2. If `resumptionSteps` is null, then return.
3. Set the `<img>`'s lazy load resumption steps to null.
4. Invoke `resumptionSteps`.

### Example

```html
<img src="1.jpeg" alt="1">
<img src="2.jpeg" loading=eager alt="2">
<img src="3.jpeg" loading=lazy alt="3">
<div id=very-large></div> <!-- Everything after this div is below the viewport -->
<img src="4.jpeg" alt="4">
<img src="5.jpeg" loading=lazy alt="5">
```
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.

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**
  The image might be a key part of the content, and there is no textual equivalent of the image available.

  **Note**
  In a conforming document, the absence of the `alt` attribute indicates that the image is a key part of the content but that a textual replacement for the image was not available when the image was generated.

  If the image is available and the user agent is configured to display that image, then the element represents the element's image data.
  
  If the image has a `src` attribute whose value is the empty string, then the element represents nothing.
  
  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 has a `title` attribute whose value is not the empty string, then return the value of that attribute.
2. 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 `flow content` descendants other than `inter-element whitespace` and the `img` element, then return the contents of the first such `figcaption` element.

3. Return nothing. (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`, `srccset` and `sizes` IDL attributes must reflect the respective content attributes of the same name.

The `crossOrigin` IDL attribute must reflect the `crossorigin` content attribute, limited to only known values.

The `useMap` IDL attribute must reflect the `usemap` content attribute.

The `isMap` IDL attribute must reflect the `ismap` content attribute.

The `referrerPolicy` IDL attribute must reflect the `referrerpolicy` content attribute, limited to only known values.

The `decoding` IDL attribute must reflect the `decoding` content attribute, limited to only known values.

The `loading` IDL attribute must reflect the `loading` content attribute, limited to only known values.

**For web developers (non-normative)**
The IDL attributes `width` and `height` must return the actual rendered dimensions of the image, or zero if the dimensions are not known. These attributes return the actual rendered dimensions of the image, or zero if the dimensions are not known.

The IDL attributes `naturalWidth` and `naturalHeight` must return the intrinsic dimensions of the image, or zero if the dimensions are not known.

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 `img` element's `current request`'s `state` is `completely available` and its `pending request` is null.
- The `img` element's `current request`'s `state` is `broken` and its `pending request` is null.

Otherwise, the attribute must return false.

The `currentSrc` IDL attribute must return the `img` element's `current request`'s `current URL`.

The `decode()` method, when invoked, must perform the following steps:

1. Let promise be a new promise.
2. Queue a microtask to perform the following steps:

The image `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]

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]

The 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.

These attributes return the intrinsic dimensions of the image, or zero if the dimensions are not known.

Returns true if the image has been completely downloaded or if no image is specified; otherwise, returns false.

Returns the image's absolute URL.

This method causes the user agent to decode the image in parallel, returning a promise that fulfills when decoding is complete.

The promise will be rejected with an "EncodingError" DOMException if the image cannot be decoded.

Returns a new `img` element, with the `width` and `height` attributes set to the values passed in the relevant arguments, if applicable.

These attributes return the intrinsic dimensions of the image, or zero if the dimensions are not known. This is done because updating the image data takes place in a microtask as well. Thus, to make code such as

```javascript
img.src = "stars.jpg";
img.decode();
```
properly decode stars.jpg, we need to delay any processing by one microtask.

1. If any of the following conditions are true about this `img` element:
   - its node document is not an active document;
   - its current request’s state is broken,

   then reject promise with an "EncodingError" DOMException.

2. Otherwise, in parallel, wait for one of the following cases to occur, and perform the corresponding actions:
   - This `img` element’s node document stops being an active document.
   - This `img` element’s current request changes or is mutated.
   - This `img` element’s current request’s state becomes broken.

     Reject promise with an "EncodingError" DOMException.

   - This `img` element’s current request’s state becomes completely available.

     Decode the image.

     If decoding does not need to be performed for this image (for example because it is a vector graphic), resolve promise with undefined.

     If decoding fails (for example due to invalid image data), reject promise with an "EncodingError" DOMException.

     If the decoding process completes successfully, resolve promise with undefined.

   User agents should ensure that the decoded media data stays readily available until at least the end of the next successful update the rendering step in the event loop. This is an important part of the API contract, and should not be broken if at all possible. (Typically, this would only be violated in low-memory situations that require evicting decoded image data, or when the image is too large to keep in decoded form for this period of time.)

   **Note**

   Animated images will become completely available only after all their frames are loaded. Thus, even though an implementation could decode the first frame before that point, the above steps will not do so, instead waiting until all frames are available.

3. Return promise.

**Example**

Without the `decode()` method, the process of loading an `img` element and then displaying it might look like the following:

```javascript
const img = new Image();
img.src = "nebula.jpg";
img.onload = () => {
  document.body.appendChild(img);
};
img.onerror = () => {
  document.body.appendChild(new Text("Could not load the nebula :("));
};
```

However, this can cause notable dropped frames, as the paint that occurs after inserting the image into the DOM causes a synchronous decode on the main thread.

This can instead be rewritten using the `decode()` method:

```javascript
const img = new Image();
img.src = "nebula.jpg";
```
A legacy factory function is provided for creating `HTMLImageElement` objects (in addition to the factory methods from DOM such as `createElement()`):

```javascript
Image(width, height). When invoked, the legacy factory function must perform the following steps:

1. Let `document` be the current global object's associated `Document`.
2. Let `img` be the result of creating an element given `document`, `img`, and the HTML namespace.
3. If `width` is given, then set an attribute value for `img` using "width" and `width`.
4. If `height` is given, then set an attribute value for `img` using "height" and `height`.
5. Return `img`.
```

This latter form avoids the dropped frames of the original, by allowing the user agent to decode the image in parallel, and only inserting it into the DOM (and thus causing it to be painted) once the decoding process is complete.

```javascript
img.decode().then(() => {
  document.body.appendChild(img);
}).catch(() => {
  document.body.appendChild(new Text("Could not load the nebula :("));
});
```

A legacy factory function is provided for creating `HTMLImageElement` objects (in addition to the factory methods from DOM such as `createElement()`): `Image(width, height)`. When invoked, the legacy factory function must perform the following steps:

1. Let `document` be the current global object's associated `Document`.
2. Let `img` be the result of creating an element given `document`, `img`, and the HTML namespace.
3. If `width` is given, then set an attribute value for `img` using "width" and `width`.
4. If `height` is given, then set an attribute value for `img` using "height" and `height`.
5. Return `img`.

```javascript
const container = document.querySelector("#container");

const { containerWidth, containerHeight } = computeDesiredSize();
requestAnimationFrame(() => {
  container.style.width = containerWidth;
  container.style.height = containerHeight;
});

// ...

const img = new Image();
img.src = "supernova.jpg";
img.decode().then(() => {
  requestAnimationFrame(() => container.appendChild(img));
});
```

A single image can have different appropriate alternative text depending on the context.

In each of the following cases, the same image is used, yet the `alt` text is different each time. The image is the coat of arms of the Carouge municipality in the canton Geneva in Switzerland.

Here it is used as a supplementary icon:

```html
<p>I lived in <img src="carouge.svg" alt="" Carouge.</p>
```

Here it is used as an icon representing the town:

```html
<p>Home town: <img src="carouge.svg" alt="Carouge"/></p>
```

Here it is used as part of a text on the town:
Carouge has a coat of arms. It is used as decoration all over the town.

The coat of arms depicts a lion, sitting in front of a tree. It is used as decoration all over the town.

She picked up the folder and a piece of paper fell out. Shaped like a shield, the paper had a red background, a green tree, and a yellow lion with its tongue hanging out and whose tail was shaped like an S. The answer she had been looking for all this time was simply the letter S! How had she not seen that before? It all came together now. The phone call where Hector had referred to a lion's tail, the time Maria had stuck her tongue out...

The last user to have uploaded a coat of arms uploaded this one: User-uploaded coat of arms.

Ideally, the author would find a way to provide real replacement text even in this case, e.g. by asking the previous user. Not providing replacement text makes the document more difficult to use for people who are unable to view images, e.g. blind users, or users with very low-bandwidth connections or who pay by the byte, or users who are forced to use a text-only web browser.

Example

Here are some more examples showing the same picture used in different contexts, with different appropriate alternate texts each time.

My cats

Fluffy is my favorite. She likes playing with a ball of yarn. She's just too cute.

My other cat, Miles just eats and sleeps.

Photography

The trick here is to know how to anticipate; to know at what speed and what distance the subject will pass by. A cat flying by, chasing a ball of yarn, can be photographed quite nicely using this technique.

Nature by night

To achieve this, you'll need either an extremely sensitive film, or immense flash lights.
**4.8.4 Images**

**4.8.4.1 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 its `src` attribute.

```html
<article>
  <h1>About me</h1>
  <h2>My pets</h2>
  <p>I've got a cat named Fluffy and a dog named Miles.</p>
  <img src="fluffy.jpg" alt="Fluffy, my cat, tends to keep itself busy."/>
  <p>My dog Miles and I like go on long walks together.</p>
  <h2>music</h2>
  <p>After our walks, having emptied my mind, I like listening to Bach.</p>
</article>

<article>
  <h1>Fluffy and the Yarn</h1>
  <p>Fluffy was a cat who liked to play with yarn. She also liked to jump.</p>
  <aside><img src="fluffy.jpg" alt="" title="Fluffy"></aside>
  <p>She would play in the morning, she would play in the evening.</p>
</article>

4.8.4 Images 5832

4.8.4.1 Introduction 5832

To embed an image in HTML, when there is only a single image resource, use the `img` element and its `src` attribute.

Example

```
<h2>From today's featured article</h2>
<img src="/uploads/100-marie-lloyd.jpg" alt="" width="100" height="150">
<p><b><a href="/wiki/Marie_Lloyd">Marie Lloyd</a> (1870–1922)</b> was an English <a href="/wiki/Music_hall">music hall</a> singer, ...</p>
```

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**

```
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
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
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
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
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
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).
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**

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 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**

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*...
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 then no image will be downloaded at all for users with scripting disabled and the aggressive image downloading optimization will also be disabled.

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 `img` 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**

```html
<h2>From today's featured article</h2>
<img src="/uploads/100-marie-lloyd.jpg"
     srcset="/uploads/150-marie-lloyd.jpg 1.5x, /uploads/200-marie-lloyd.jpg 2x"
     alt="" width="100" height="150">
<p>(1870–1922) was an English singer, ...</p>
```

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 `img` 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**

In this example, a banner image takes up the entire viewport width (using appropriate CSS).

```html
<h1><img sizes="100vw" srcset="wolf-400.jpg 400w, wolf-800.jpg 800w, wolf-1600.jpg 1600w"
         src="wolf-400.jpg" alt="The rad wolf"></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 1.25w, wolf-800.jpg 2.5w, wolf-1600.jpg 5w`. On the other hand, if the user's screen is 1200 CSS pixels wide, this is equivalent to specifying `wolf-400.jpg 0.33w, wolf-800.jpg 0.67w, wolf-1600.jpg 1.33w`. 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.

### Example

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.

```html
<img sizes="(max-width: 30em) 100vw, (max-width: 50em) 50vw, calc(33vw - 100px)"
     srcset="swing-200.jpg 200w, swing-400.jpg 400w, swing-800.jpg 800w, swing-1600.jpg 1600w"
     src="swing-400.jpg" alt="Kettlebell Swing">
```

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).

```html
<picture>
  <source media="(min-width: 45em)" srcset="large.jpg">
  <source media="(min-width: 32em)" srcset="med.jpg">
  <img src="small.jpg" alt="The wolf runs through the snow.">
</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.

```css
img { width: 300px; height: 300px }
@media (min-width: 32em) { img { width: 500px; height:300px } }
@media (min-width: 45em) { img { width: 700px; height:400px } }
```

### Example

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.

```html
<h1>
  <picture>
    <source media="(max-width: 500px)" srcset="banner-phone.jpeg, banner-phone-HD.jpeg 2x">
    <img src="banner.jpeg" srcset="banner-HD.jpeg 2x" alt="The Breakfast Combo">
  </picture>
</h1>
```
The `type` attribute on the `source` element can be used, to provide multiple images in different formats.

### Example

```html
<h2>From today's featured article</h2>
<p><a href="/wiki/Marie_Lloyd">Marie Lloyd</a> (1870–1922) was an English singer, ...</p>
</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 `img` element will be chosen.

### 4.8.4.1.1 Adaptive images

This section is non-normative.

CSS and media queries can be used to construct graphical page layouts that adapt dynamically to the user's environment, in particular to different viewport dimensions and pixel densities. For content, however, CSS does not help; instead, we have the `img` element's `srcset` attribute and the `picture` element. This section walks through a sample case showing how to use these features.

Consider a situation where on wide screens (wider than 600 CSS pixels) a 300×150 image named `a-rectangle.png` is to be used, but on smaller screens (600 CSS pixels and less), a smaller 100×100 image called `a-square.png` is to be used. The markup for this would look like this:

```html
<figure>
  <picture>
    <source srcset="/uploads/100-marie-lloyd.webp" type="image/webp"/>
    <source srcset="/uploads/100-marie-lloyd.jxr" type="image/vnd.ms-photo"/>
    <img src="/uploads/100-marie-lloyd.jpg" alt="Barney Frank wears a suit and glasses."/>
  </picture>
  <figcaption>Barney Frank, 2011</figcaption>
</figure>
```

For details on what to put in the `alt` attribute, see the Requirements for providing text to act as an alternative for images section.

The problem with this is that the user agent does not necessarily know what dimensions to use for the image when the image is loading. To avoid the layout having to be reflowed multiple times as the page is loading, CSS and CSS media queries can be used to provide the dimensions:

```html
<style>
  #a { width: 300px; height: 150px; }
  @media (max-width: 600px) { #a { width: 100px; height: 100px; } }
</style>
<figure>
  <picture>
    <source srcset="/uploads/100-marie-lloyd.png" media="(max-width: 600px)"/>
    <img src="/uploads/100-marie-lloyd.jpg" alt="Barney Frank wears a suit and glasses." id="a"/>
  </picture>
  <figcaption>Barney Frank, 2011</figcaption>
</figure>
```
Alternatively, the `width` and `height` attributes can be used to provide the width and height for legacy user agents, using CSS just for the user agents that support `picture`:

```html
<figure>
  <picture>
    <source srcset="a-square.png" media="(max-width: 600px)"/>
    <source srcset="a-rectangle.png" width="300" height="150">
    <img src="a-rectangle.png" alt="Barney Frank wears a suit and glasses." id="a">
  </picture>
  <figcaption>Barney Frank, 2011</figcaption>
</figure>
```

The `img` element is used with the `src` attribute, which gives the URL of the image to use for legacy user agents that do not support the `picture` element. This leads to a question of which image to provide in the `src` attribute.

If the author wants the biggest image in legacy user agents, the markup could be as follows:

```html
<picture>
  <source srcset="pear-mobile.jpeg" media="(max-width: 720px)"/>
  <source srcset="pear-tablet.jpeg" media="(max-width: 1280px)"/>
  <img src="pear-mobile.jpeg" alt="The pear is juicy.">
</picture>
```

However, if legacy mobile user agents are more important, one can list all three images in the `source` elements, overriding the `src` attribute entirely.

```html
<picture>
  <source srcset="pear-mobile.jpeg" media="(max-width: 720px)"/>
  <source srcset="pear-tablet.jpeg" media="(max-width: 1280px)"/>
  <source srcset="pear-desktop.jpeg">
  <img src="pear-mobile.jpeg" alt="The pear is juicy.">
</picture>
```

Since at this point the `src` attribute is actually being ignored entirely by `picture`-supporting user agents, the `src` attribute can default to any image, including one that is neither the smallest nor biggest:

```html
<picture>
  <source srcset="pear-mobile.jpeg" media="(max-width: 720px)"/>
  <source srcset="pear-tablet.jpeg" media="(max-width: 1280px)"/>
  <source srcset="pear-desktop.jpeg">
  <img src="pear-tablet.jpeg" alt="The pear is juicy.">
</picture>
```

Above the `max-width` media feature is used, giving the maximum (viewport) dimensions that an image is intended for. It is also possible to use `min-width` instead.

```html
<picture>
  <source srcset="pear-desktop.jpeg" media="(min-width: 1281px)"/>
  <source srcset="pear-tablet.jpeg" media="(min-width: 721px)"/>
  <img src="pear-mobile.jpeg" alt="The pear is juicy.">
</picture>
```
4.8.4.2 Attributes common to source, img, and link elements

4.8.4.2.1 Srcset attributes

A srcset attribute is an attribute with requirements defined in this section.

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 ASCII whitespace after the URL, the following image candidate string, if there is one, must begin with one or more ASCII whitespace.

An image candidate string consists of the following components, in order, with the further restrictions described below this list:

1. Zero or more ASCII whitespace.
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 ASCII whitespace.
4. Zero or one of the following:
   ◦ A width descriptor, consisting of: ASCII whitespace, 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: ASCII whitespace, 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 ASCII whitespace.

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 an image candidate string for an element has the width descriptor specified, all other image candidate strings for that element must also have the width descriptor specified.

If an element has a sizes attribute present, all image candidate strings for that element must have the width descriptor specified.

4.8.4.2.2 Sizes attributes

A sizes attribute is an attribute with requirements defined in this section.

If present, the value must be a valid source size list.

A valid source size list is a string that matches the following grammar: [CSSVALUES][MQ]

\[
<\text{source-size-list}> = [ <\text{source-size}># , ]? <\text{source-size-value}>
\]

\[
<\text{source-size}> = <\text{media-condition}> <\text{source-size-value}>
\]

\[
<\text{source-size-value}> = <\text{length}>
\]

A source-size-value must not be negative, and must not use CSS functions other than the math functions.

The source-size-value gives the intended layout width of the image. The author can specify different widths for different environments with media-conditions.
An `<img>` 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.

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's state is either partially available or completely available, the image request is said to be available.

When an `img` element's current request's state is completely available and the user agent can decode the media data without errors, then the `img` element is said to be fully decodable.

An image request's state is initially unavailable.

When an `img` element's current request is available, the `img` element 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.

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.

Each `img` 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 `img` 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

For example, if the current pixel density is 3.125, that means that there are 300 device pixels per CSS inch, and thus if the image data is 300x600, it has intrinsic dimensions of 96 CSS pixels by 192 CSS pixels.
All `<img>` and `<link>` elements are 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 pixel 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 `<img>` element's node document's viewport. Other units must be interpreted the same as in Media Queries.

A parse error for algorithms in this section indicates a non-fatal mismatch between input and requirements. User agents are encouraged to expose parse errors somehow.

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 `<img>` element (e.g. XML files whose document 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 honour any animation in the resource.

This specification does not specify which image types are to be supported.

### 4.8.4.3.1 When to obtain images

By default, images are obtained immediately. User agents may provide users with the option to instead obtain them on-demand. (The on-demand option might be used by bandwidth-constrained users, for example.)

When obtaining images immediately, the user agent must synchronously update the image data of the `<img>` element, with the restart animation flag set if so stated, whenever that element is created or has experienced relevant mutations.

When obtaining images on demand, the user agent must update the image data of an `<img>` element whenever it needs the image data (i.e., on demand), but only if the `<img>` element's current request's state is unavailable. When an `<img>` element has experienced relevant mutations, if the user agent only obtains images on demand, the `<img>` element's current request's state must return to unavailable.

### 4.8.4.3.2 Reacting to DOM mutations

The relevant mutations for an `<img>` 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's `referrerpolicy` 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.

### 4.8.4.3.3 The list of available images

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 `<Document>`s), 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.

**Note**

The user agent can also store the image data separately from the list of available images.

**Example**

For example, if a resource has the HTTP response header `Cache-Control: must-revalidate`, and its ignore higher-layer caching flag is unset, 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 304 Not Modified status.

### 4.8.4.3.4 Decoding images

Image data is usually encoded in order to reduce file size. This means that in order for the user agent to present the image to the screen, the data needs to be decoded. Decoding is the process which converts an image’s media data into a bitmap form, suitable for presentation to the screen. Note that this process can be slow relative to other processes involved in presenting content. Thus, the user agent can choose when to perform decoding, in order to create the best user experience.

Image decoding is said to be synchronous if it prevents presentation of other content until it is finished. Typically, this has an effect of atomically presenting the image and any other content at the same time. However, this presentation is delayed by the amount of time it takes to perform the decode.

Image decoding is said to be asynchronous if it does not prevent presentation of other content. This has an effect of presenting non-image content faster. However, the image content is missing on screen until the decode finishes. Once the decode is finished, the screen is updated with the image.

In both synchronous and asynchronous decoding modes, the final content is presented to screen after the same amount of time has elapsed. The main difference is whether the user agent presents non-image content ahead of presenting the final content.

In order to aid the user agent in deciding whether to perform synchronous or asynchronous decode, the decoding attribute can be set on `<img>` elements. The possible values of the decoding attribute are the following image decoding hint keywords:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sync</td>
<td>Sync</td>
<td>Indicates a preference to decode this image synchronously for atomic presentation with other content.</td>
</tr>
<tr>
<td>async</td>
<td>Async</td>
<td>Indicates a preference to decode this image asynchronously to avoid delaying presentation of other content.</td>
</tr>
<tr>
<td>auto</td>
<td>Auto</td>
<td>Indicates no preference in decoding mode (the default).</td>
</tr>
</tbody>
</table>
When decoding an image, the user agent should respect the preference indicated by the `decoding` attribute’s state. If the state indicated is `auto`, then the user agent is free to choose any decoding behavior.

**Note**

It is also possible to control the decoding behavior using the `decode()` method. Since the `decode()` method performs decoding independently from the process responsible for presenting content to screen, it is unaffected by the `decoding` attribute.

### 4.8.4.3.5 Updating the image data

**Note**

This algorithm cannot be called from steps running in parallel. If a user agent needs to call this algorithm from steps running in parallel, it needs to queue a task to do so.

When the user agent is to update the image data of an `img` 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:
   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 `img` element was started after this instance (even if it aborted and is no longer running), then return.
   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’s state to unavailable, set pending request to null, and return.
3. Let selected source be null and selected pixel density be undefined.
4. If the element does not use `srcset` or `picture`, and it has a `src` attribute specified whose value is not the empty string, then set selected source to the value of the element’s `src` attribute and set selected pixel density to 1.0.
5. Set the element’s last selected source to selected source.
6. If selected source is not null, then:
   1. Parse selected source, relative to the element’s `node document`. If that is not successful, then abort this inner set of steps. Otherwise, let `urlString` be the resulting URL string.
   2. Let key be a tuple consisting of `urlString`, 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, then:
      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. Set pending request to null.
      4. Let current request be a new image request whose `image data` is that of the entry and whose state is completely available.
      5. Update the presentation of the image appropriately.
      6. Set current request’s current pixel density to selected pixel density.
      7. Queue an element task on the DOM manipulation task source given the `img` element and following steps:
1. If restart animation is set, then restart the animation.

2. Set current request’s current URL to urlString.

3. Fire an event named load at the img element.

8. Abort the update the image data algorithm.

7. Queue a microtask to perform the rest of this algorithm, allowing the task that invoked this algorithm to continue.

8. 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 return.

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.

9. Let selected source and selected pixel density be the URL and pixel density that results from selecting an image source, respectively.

10. If selected source is null, then:

1. Set the current request’s state to broken, abort the image request for the current request and the pending request, and set pending request to null.

2. Queue an element task on the DOM manipulation task source given the img element and the following steps:

   1. Change the current request’s current URL to the empty string.

   2. If the element has a src attribute or it uses srcset or picture, fire an event named error at the img element.

3. Return.

11. Parse selected source, relative to the element’s node document, and let urlString be the resulting URL string. If that is not successful, then:

1. Abort the image request for the current request and the pending request.

2. Set the current request’s state to broken.

3. Set pending request to null.

4. Queue an element task on the DOM manipulation task source given the img element and the following steps:

   1. Change the current request’s current URL to selected source.

   2. Fire an event named error at the img element.

5. Return.

12. If the pending request is not null and urlString is the same as the pending request’s current URL, then return.

13. If urlString is the same as the current request’s current URL and current request’s state is partially available, then abort the image request for the pending request, queue an element task on the DOM manipulation task source given the img element to restart the animation if restart animation is set, and return.

14. If the pending request is not null, then abort the image request for the pending request.

15. Set image request to a new image request whose current URL is urlString.

16. If current request’s state is unavailable or broken, then set the current request to image request. Otherwise, set the pending request to image request.

17. Let request be the result of creating a potential-CORS request given urlString, “image”, and the current state of the element’s crossorigin content attribute.

18. Set request’s client to the element’s node document’s relevant settings object.
19. If the element uses srcset or picture, set request's initiator to "imageset".

20. Set request's referer policy to the current state of the element's refererpolicy attribute.

21. Let delay load event be true if the img's lazy loading attribute is in the Eager state, or if scripting is disabled for the img element, and false otherwise.

22. If the will lazy load element steps given the img return true, then:
   1. Set the img's lazy load resumption steps to the rest of this algorithm starting with the step labeled fetch the image.
   2. Start intersection-observing a lazy loading element for the img element.
   3. Return.

23. Fetch the image: 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).

When delay load event is true, 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.

24. Continue the remaining steps in parallel, but without missing any data from fetching.

25. 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, its state is unavailable, 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, its state is unavailable, 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's current request's state to completely available and queue an element task on the DOM manipulation task source given the img element to fire an event named load at the img element.

→ 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\(^{p337}\), set image request's state\(^{p337}\) to partially available\(^{p337}\).

2. Otherwise, if the user agent is able to determine image request's image's width and height, and image request is current request\(^{p337}\), update the img\(^{p331}\)'s element's presentation appropriately and set image request's state\(^{p337}\) to partially available\(^{p337}\).

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\(^{p337}\), abort the image request\(^{p337}\) for the current request\(^{p337}\), and the pending request\(^{p337}\), upgrade the pending request to the current request\(^{p343}\), set current request\(^{p337}\)'s state\(^{p337}\) to broken\(^{p337}\), and fire an event named error\(^{p1266}\) at the img\(^{p321}\) element.

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\(^{p337}\), abort the image request\(^{p337}\) for image request and fire an event named error\(^{p1266}\) at the img\(^{p321}\) element.

That task\(^{p391}\), and each subsequent task\(^{p391}\), that is queued\(^{p932}\) by the networking task source\(^{p938}\) while the image is being fetched, if image request is the current request\(^{p337}\), 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\(^{p391}\) that is queued\(^{p932}\) by the networking task source\(^{p938}\) once the resource has been fetched must additionally run these steps:

1. If image request is the pending request\(^{p337}\), abort the image request\(^{p341}\) for the current request\(^{p337}\), upgrade the pending request to the current request\(^{p343}\) and update the img\(^{p321}\) element's presentation appropriately.
2. Set image request to the completely available\(^{p337}\) state.
3. Add the image to the list of available images\(^{p339}\), using the key key, with the ignore higher-layer caching\(^{p339}\) flag set.
4. Fire an event named load\(^{p1266}\) at the img\(^{p321}\) element.

Otherwise

The image data is not in a supported file format; the user agent must set image request's state\(^{p337}\) to broken\(^{p337}\), abort the image request\(^{p343}\) for the current request\(^{p337}\), and the pending request\(^{p337}\), upgrade the pending request to the current request\(^{p343}\) if image request is the pending request\(^{p337}\), and then queue an element task\(^{p932}\) on the DOM manipulation task source\(^{p938}\) given the img\(^{p321}\) element to fire an event named error\(^{p1266}\) at the img\(^{p321}\) element.

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 connected.

To abort the image request for an image request\(^{p337}\) image request means to run the following steps:

1. Forget image request's image data\(^{p337}\), 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 img\(^{p321}\) element means to run the following steps:

1. Let the img\(^{p321}\) element's current request\(^{p337}\) be the pending request\(^{p337}\).
2. Let the img\(^{p321}\) element's pending request\(^{p337}\) be null.

### 4.8.4.3.6 Selecting an image source\(^{p344}\)

When asked to select an image source\(^{p344}\) for a given img\(^{p321}\) or link\(^{p128}\) element el, user agents must do the following:

1. Update the source set\(^{p344}\) for el.
2. If el's source set\(^{p338}\) is empty, return null as the URL and undefined as the pixel density.
3. Otherwise, take el's source set\(^{p338}\) and let it be source set.
4. If an entry b in source set has the same associated pixel density descriptor\(^{p336}\) 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 pixel 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.

4.8.4.3.7 Updating the source set

When asked to update the source set for a given img or link element el, user agents must do the following:

1. Set el's source set to an empty source set.
2. Let elements be « el ».
3. If el is an img element whose parent node is a picture element, then replace the contents of elements with el's parent node's child elements, retaining relative order.
4. For each child in elements:
   1. If child is el:
      1. Let source set be an empty source set.
      2. If child has a srcset or imagesrcset attribute, parse child's srcset attribute and set source set to the returned source set.
      3. Parse child's sizes attribute, and let source set's source size be the returned value.
      4. If child has a src or href attribute whose value is not the empty string and source set does not contain an image source with a pixel density descriptor value of 1, and no image source with a width descriptor, append child's src or href attribute value to source set.
      5. Normalize the source densities of source set.
      7. Return.

   2. Assert: child is not a link element.
   3. If child is not a source element, continue to the next child. Otherwise, child is a source element.
   4. If child does not have a srcset attribute, continue to the next child.
   5. Parse child's srcset attribute and let the returned source set be source set.
   6. If source set has zero image sources, continue to the next child.
   7. If child has a media attribute, and its value does not match the environment, continue to the next child.
   8. Parse child's sizes attribute with the fallback width width, and let source set's source size be the returned value.
   9. If child has a type attribute, and its value is an unknown or unsupported MIME type, continue to the next child.
   10. Normalize the source densities of source set.
   11. Let el's source set be source set.
   12. Return.

Note: If el is a link element, then elements contains only el, so this step will be reached immediately and the rest of the algorithm will not run.
4.8.4.3.8 Parsing a srcset attribute

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 code points that are ASCII whitespace or U+002C COMMA characters from input given position. If any U+002C COMMA characters were collected, that is a parse error.
5. If position is past the end of input, return candidates.
6. Collect a sequence of code points that are not ASCII whitespace from input given position, and let that be url.
7. Let descriptors be a new empty list.
8. If url ends with U+002C (,), then:
   1. Remove all trailing U+002C COMMA characters from url. If this removed more than one character, that is a parse error.
   Otherwise:
      1. Descriptor tokenizer: Skip ASCII whitespace within input given position.
      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:
         ↪ ASCII whitespace
         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 PARENTHESES ()
         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 PARENTHESES ()
         Append c to current descriptor. Set state to in descriptor.
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:

ASCII whitespace
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 step.

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\textsuperscript{p69} to the descriptor. If the result is zero, let \textit{error} be yes. Otherwise, let \textit{future-compat-h} be the result.

→ Anything else

Let \textit{error} be yes.

14. If \textit{future-compat-h} is not \textit{absent} and \textit{width} is \textit{absent}, let \textit{error} be yes.

15. If \textit{error} is still no, then append a new \textit{image source}\textsuperscript{p338} to \textit{candidates} whose URL is \textit{url}, associated with a width \textit{width} if not \textit{absent} and a pixel density \textit{density} if not \textit{absent}. Otherwise, there is a \textit{parse error}\textsuperscript{p338}.

16. Return to the step labeled \textit{splitting loop}.

\subsection*{4.8.4.3.9 Parsing a sizes attribute} \textsuperscript{p34
7}

When asked to \textit{parse a sizes attribute} from an element, \textit{parse a comma-separated list of component values} from the value of the element’s \textit{sizes attribute}\textsuperscript{p336} (or the empty string, if the attribute is absent), and let \textit{unparsed sizes list} be the result. \textsuperscript{[CSSSYNTAX]31272}

For each \textit{unparsed size} in \textit{unparsed sizes list}:

1. Remove all consecutive \textit{<whitespace-token>}s from the end of \textit{unparsed size}. If \textit{unparsed size} is now empty, that is a \textit{parse error}\textsuperscript{p338}; continue to the next iteration of this algorithm.

2. If the last \textit{component value} in \textit{unparsed size} is a valid non-negative \textit{<source-size-value>}\textsuperscript{p336}, let \textit{size} be its value and remove the \textit{component value} from \textit{unparsed size}. Any CSS function other than the \textit{math functions} is invalid. Otherwise, there is a \textit{parse error}\textsuperscript{p338}; continue to the next iteration of this algorithm.

3. Remove all consecutive \textit{<whitespace-token>}s from the end of \textit{unparsed size}. If \textit{unparsed size} is now empty, return \textit{size} and exit this algorithm. If this was not the last item in \textit{unparsed sizes list}, that is a \textit{parse error}\textsuperscript{p338}.

4. Parse the remaining \textit{component values} in \textit{unparsed size} as a \textit{<media-condition>}. If it does not parse correctly, or it does parse correctly but the \textit{<media-condition>} evaluates to false, continue to the next iteration of this algorithm. \textsuperscript{[MQ]11274}

5. Return \textit{size} and exit this algorithm.

If the above algorithm exhausts \textit{unparsed sizes list} without returning a \textit{size} value, then return 100vw.

\begin{note}
While a \textit{valid source size list}\textsuperscript{p336} only contains a bare \textit{<source-size-value>}\textsuperscript{p336} (without an accompanying \textit{<media-condition>}) as the last entry in the \textit{<source-size-list>}\textsuperscript{p336}, 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.
\end{note}

\subsection*{4.8.4.3.10 Normalizing the source densities} \textsuperscript{p34
7}

An \textit{image source}\textsuperscript{p338} can have a \textit{pixel density descriptor}\textsuperscript{p336}, a \textit{width descriptor}\textsuperscript{p336}, or no descriptor at all accompanying its URL. Normalizing a \textit{source set}\textsuperscript{p338} gives every \textit{image source}\textsuperscript{p338} a \textit{pixel density descriptor}\textsuperscript{p336}.

When asked to \textit{normalize the source densities} of a \textit{source set}\textsuperscript{p338} \textit{source set}, the user agent must do the following:

1. Let source size be \textit{source set}’s \textit{source size}\textsuperscript{p338}.

2. For each \textit{image source}\textsuperscript{p338} in \textit{source set}:

   1. If the \textit{image source}\textsuperscript{p338} has a \textit{pixel density descriptor}\textsuperscript{p336}, continue to the next \textit{image source}\textsuperscript{p338}.

   2. Otherwise, if the \textit{image source}\textsuperscript{p338} has a \textit{width descriptor}\textsuperscript{p336}, replace the \textit{width descriptor}\textsuperscript{p336} with a \textit{pixel density descriptor}\textsuperscript{p336} with a value\textsuperscript{p336} of the \textit{width descriptor value}\textsuperscript{p336} divided by the \textit{source size}\textsuperscript{p338} and a unit of \textit{x}.

\begin{note}
If the \textit{source size}\textsuperscript{p338} is zero, the density would be infinity, which results in the \textit{intrinsic dimensions} being zero.
\end{note}
3. Otherwise, give the image source a pixel density descriptor of 1x.

4.8.4.3.11 Reacting to environment changes

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 img 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. 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 img 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 return.

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 return.

5. ⌛ If selected source and selected pixel density are the same as the element's last selected source and current pixel density, then return.

6. ⌛ Parse selected source, relative to the element's node document, and let urlString be the resulting URL string. If that is not successful, then return.

7. ⌛ Let corsAttributeState be the state of the element's crossorigin content attribute.

8. ⌛ Let origin be the img element's node document's origin.

9. ⌛ Let client be the img element's node document's relevant settings object.

10. ⌛ Let key be a tuple consisting of urlString, corsAttributeState, and, if corsAttributeState is not No CORS, origin.

11. ⌛ Let image request be a new image request whose current URL is urlString

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:

1. Let request be the result of creating a potential-CORS request given urlString, "image", and corsAttributeState.

2. Set request's client to client, initiator to "imageset", and set request's synchronous flag.

3. Set request's referer policy to the current state of the element's refererpolicy attribute.

4. Let response be the result of fetching request.

5. 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.
6. 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 an element task on the DOM manipulation task source given the img element and the following steps:

1. If the img element has experienced relevant mutations since this algorithm started, then let pending request be null and abort these steps.
2. Let the img element's last selected source be selected source and the img element's current pixel density be selected pixel density.
3. Set the image request's state to completely available.
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 an event named load at the img element.

4.8.4.4 Requirements for providing text to act as an alternative for images

4.8.4.4.1 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 replacement for the image. The specific requirements for the alt attribute depend on what the image is intended to represent, as described in the following sections.

The most general rule to consider when writing alternative text is the following: the intent is that replacing every image with the text of its alt attribute does not change the meaning of the page.

So, in general, alternative text can be written by considering what one would have written had one not been able to include the image.

A corollary to this is that the alt attribute's value should never contain text that could be considered the image's caption, title, or legend. It is supposed to contain replacement text that could be used by users instead of the image; it is not meant to supplement the image. The title attribute can be used for supplemental information.

Another corollary is that the alt attribute's value should not repeat information that is already provided in the prose next to the image.

Note

One way to think of alternative text is to think about how you would read the page containing the image to someone over the phone, without mentioning that there is an image present. Whatever you say instead of the image is typically a good start for writing the alternative text.

4.8.4.4.2 A link or button containing nothing but the image

When an a element that creates a hyperlink, or a button element, has no textual content but contains one or more images, the alt attributes must contain text that together convey the purpose of the link or button.

Example

In this example, a user is asked to pick their preferred color from a list of three. Each color is given by an image, but for users who have configured their user agent not to display images, the color names are used instead:

```html
<h1>Pick your color</h1>
<ul>
  <li><a href="green.html">img src="green.jpeg" alt="Green"></a></li>
  <li><a href="blue.html">img src="blue.jpeg" alt="Blue"></a></li>
  <li><a href="red.html">img src="red.jpeg" alt="Red"></a></li>
</ul>
```
Sometimes something can be more clearly stated in graphical form, for example as a flowchart, a diagram, a graph, or a simple map showing directions. In such cases, an image can be given using the `<img>` element, but the lesser textual version must still be given, so that users who are unable to view the image (e.g. because they have a very slow connection, or because they are using a text-only browser, or because they are listening to the page being read out by a hands-free automobile voice web browser, or simply because they are blind) are still able to understand the message being conveyed.

The text must be given in the `alt` attribute, and must convey the same message as the image specified in the `src` attribute.

It is important to realize that the alternative text is a replacement for the image, not a description of the image.

In this example, each button has a set of images to indicate the kind of color output desired by the user. The first image is used in each case to give the alternative text.

```html
<button name="rgb"><img src="red" alt="RGB"><img src="green" alt=""><img src="blue" alt=""></button>

<button name="cmyk"><img src="cyan" alt="CMYK"><img src="magenta" alt=""><img src="yellow" alt=""><img src="black" alt=""></button>
```

Since each image represents one part of the text, it could also be written like this:

```html
<button name="rgb"><img src="red" alt="R"><img src="green" alt="G"><img src="blue" alt=""></button>

<button name="cmyk"><img src="cyan" alt="C"><img src="magenta" alt="M"><img src="yellow" alt=""><img src="black" alt=""></button>
```

However, with other alternative text, this might not work, and putting all the alternative text into one image in each case might make more sense:

```html
<button name="rgb"><img src="red" alt="sRGB profile"><img src="green" alt=""><img src="blue" alt=""></button>

<button name="cmyk"><img src="cyan" alt="CMYK profile"><img src="magenta" alt=""><img src="yellow" alt=""><img src="black" alt=""></button>
```

## 4.8.4.4.3 A phrase or paragraph with an alternative graphical representation: charts, diagrams, graphs, maps, illustrations

Sometimes something can be more clearly stated in graphical form, for example as a flowchart, a diagram, a graph, or a simple map showing directions. In such cases, an image can be given using the `img` element, but the lesser textual version must still be given, so that users who are unable to view the image (e.g. because they have a very slow connection, or because they are using a text-only browser, or because they are listening to the page being read out by a hands-free automobile voice web browser, or simply because they are blind) are still able to understand the message being conveyed.

The text must be given in the `alt` attribute, and must convey the same message as the image specified in the `src` attribute.

It is important to realize that the alternative text is a replacement for the image, not a description of the image.

In the following example we have a flowchart in image form, with text in the `alt` attribute rephrasing the flowchart in prose form:

```html
<p>In the common case, the data handled by the tokenization stage comes from the network, but it can also come from script.</p>
<p><img src="images/parsing-model-overview.svg" alt="The Network passes data to the Input Stream Preprocessor, which passes it to the Tokenizer, which passes it to the Tree Construction stage. From there, data goes to both the DOM and to Script Execution. Script Execution is linked to the DOM, and, using document.write(), passes data to the Tokenizer."></p>
```

Here's another example, showing a good solution and a bad solution to the problem of including an image in a description.
First, here's the good solution. This sample shows how the alternative text should just be what you would have put in the prose if the image had never existed.

```html
<!-- This is the correct way to do things. -->
<p>You are standing in an open field west of a house.
   <img src="house.jpeg" alt="The house is white, with a boarded front door.">
   There is a small mailbox here.
</p>
```

Second, here's the bad solution. In this incorrect way of doing things, the alternative text is simply a description of the image, instead of a textual replacement for the image. It's bad because when the image isn't shown, the text doesn't flow as well as in the first example.

```html
<!-- This is the wrong way to do things. -->
<p>You are standing in an open field west of a house.
   <img src="house.jpeg" alt="A white house, with a boarded front door.">
   There is a small mailbox here.
</p>
```

Text such as "Photo of white house with boarded door" would be equally bad alternative text (though it could be suitable for the title attribute or in the figcaption element of a figure with this image).

### 4.8.4.4.4 A short phrase or label with an alternative graphical representation: icons, logos

A document can contain information in iconic form. The icon is intended to help users of visual browsers to recognize features at a glance.

In some cases, the icon is supplemental to a text label conveying the same meaning. In those cases, the alt attribute must be present but must be empty.

**Example**

Here the icons are next to text that conveys the same meaning, so they have an empty alt attribute:

```html
<nav>
   <p><a href="/help/">Help</a></p>
   <p><a href="/configure/">Configuration Tools</a></p>
</nav>
```

In other cases, the icon has no text next to it describing what it means; the icon is supposed to be self-explanatory. In those cases, an equivalent textual label must be given in the alt attribute.

**Example**

Here, posts on a news site are labeled with an icon indicating their topic.

```html
<body>
   <article>
      <header>
         <h1>Ratatouille wins Best Movie of the Year award</h1>
      </header>
      <p><img src="movies.png" alt="Movies"/>
      </p>
   </article>
   <p>Pixar has won yet another Best Movie of the Year award, making this its 8th win in the last 12 years.</p>
</body>
```
Many pages include logos, insignia, flags, or emblems, which stand for a particular entity such as a company, organization, project, band, software package, country, or some such.

If the logo is being used to represent the entity, e.g. as a page heading, the alt attribute must contain the name of the entity being represented by the logo. The alt attribute must not contain text like the word "logo", as it is not the fact that it is a logo that is being conveyed, it's the entity itself.

If the logo is being used next to the name of the entity that it represents, then the logo is supplemental, and its alt attribute must instead be empty.

If the logo is merely used as decorative material (as branding, or, for example, as a side image in an article that mentions the entity to which the logo belongs), then the entry below on purely decorative images applies. If the logo is actually being discussed, then it is being used as a phrase or paragraph (the description of the logo) with an alternative graphical representation (the logo itself), and the first entry above applies.

Example

In the following snippets, all four of the above cases are present. First, we see a logo used to represent a company:

```html
<h1><img src="XYZ.gif" alt="The XYZ company"></h1>
```

Next, we see a paragraph which uses a logo right next to the company name, and so doesn't have any alternative text:

```html
<article>
<h2>News</h2>
<p>We have recently been looking at buying the <img src="alpha.gif" alt=""> ΑΒΓ company, a small Greek company specializing in our type of product.</p>
</article>
```

In this third snippet, we have a logo being used in an aside, as part of the larger article discussing the acquisition:

```html
<aside><p><img src="alpha-large.gif" alt=""></p></aside>
<p>The ΑΒΓ company has had a good quarter, and our pie chart studies of their accounts suggest a much bigger blue slice than its green and orange slices, which is always a good sign.</p>
</article>
```

Finally, we have an opinion piece talking about a logo, and the logo is therefore described in detail in the alternative text.

```html
<p>Consider for a moment their logo:</p>
<p><img src="/images/logo" alt="It consists of a green circle with a green question mark centered inside it."/></p>
<p>How unoriginal can you get? I mean, ooooooh, a question mark, how <em>revolutionary</em>, how utterly <em>ground-breaking</em>, I'm
4.8.4.4.5 Text that has been rendered to a graphic for typographical effect

Sometimes, an image just consists of text, and the purpose of the image is not to highlight the actual typographic effects used to render the text, but just to convey the text itself.

In such cases, the alt attribute must be present but must consist of the same text as written in the image itself.

Example
Consider a graphic containing the text "Earth Day", but with the letters all decorated with flowers and plants. If the text is merely being used as a heading, to spice up the page for graphical users, then the correct alternative text is just the same text "Earth Day", and no mention need be made of the decorations:

```html
<h1><img src="earthdayheading.png" alt="Earth Day"></h1>
```

Example
An illuminated manuscript might use graphics for some of its images. The alternative text in such a situation is just the character that the image represents.

```html
<p><img src="initials/o.svg" alt="O">nce upon a time and a long long time ago, late at night, when it was dark, over the hills, through the woods, across a great ocean, in a land far away, in a small house, on a hill, under a full moon...
```

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
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.

```html
<p>Only <img src="euro.png" alt="euro">5.99!
```

An image should not be used if characters would serve an identical purpose. Only when the text cannot be directly represented using text, e.g., because of decorations or because there is no appropriate character (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.

4.8.4.4.6 A graphical representation of some of the surrounding text

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

A flowchart that repeats the previous paragraph in graphical form:

```
<p>The Network passes data to the Input Stream Preprocessor, which
passes it to the Tokenizer, which passes it to the Tree Construction
stage. From there, data goes to both the DOM and to Script Execution.
Script Execution is linked to the DOM, and, using document.write(),
passes data to the Tokenizer.</p>
```

```
<p><img src="images/parsing-model-overview.svg" alt=""></p>
```

In these cases, it would be wrong to include alternative text that consists of just a caption. If a caption is to be included, then either the `title` attribute can be used, or the `figure` and `figcaption` elements can be used. In the latter case, the image would in fact be a phrase or paragraph with an alternative graphical representation, and would thus require alternative text.

```
<!-- Using the title="" attribute -->
<p>The Network passes data to the Input Stream Preprocessor, which
passes it to the Tokenizer, which passes it to the Tree Construction
stage. From there, data goes to both the DOM and to Script Execution.
Script Execution is linked to the DOM, and, using document.write(),
passes data to the Tokenizer.</p>
```

```
<p><img src="images/parsing-model-overview.svg" alt=" title="Flowchart representation of the parsing model.""></p>
```

```
<!-- Using <figure> and <figcaption> -->
<p>The Network passes data to the Input Stream Preprocessor, which
passes it to the Tokenizer, which passes it to the Tree Construction
stage. From there, data goes to both the DOM and to Script Execution.
Script Execution is linked to the DOM, and, using document.write(),
passes data to the Tokenizer.</p>
```

```
<figure>
<img src="images/parsing-model-overview.svg" alt="The Network leads to
the Input Stream Preprocessor, which leads to the Tokenizer, which
leads to the Tree Construction stage. The Tree Construction stage
leads to two items. The first is Script Execution, which leads via
document.write() back to the Tokenizer. The second item from which
Tree Construction leads is the DOM. The DOM is related to the Script
Execution."
>
<figcaption>Flowchart representation of the parsing model.</figcaption>
</figure>
```

```
<!-- This is WRONG. Do not do this. Instead, do what the above examples do. -->
<p>The Network passes data to the Input Stream Preprocessor, which
passes it to the Tokenizer, which passes it to the Tree Construction
stage. From there, data goes to both the DOM and to Script Execution.
Script Execution is linked to the DOM, and, using document.write(),
passes data to the Tokenizer.</p>
```

```
<p><img src="images/parsing-model-overview.svg" alt="Flowchart representation of the parsing model."'></p>
```

```
<!-- Never put the image's caption in the alt="" attribute! -->
```

Example

A graph that repeats the previous paragraph in graphical form:

```
<p>According to a study covering several billion pages,
about 62% of documents on the web in 2007 triggered the Quirks
rendering mode of web browsers, about 30% triggered the Almost
Standards mode, and about 9% triggered the Standards mode.</p>
```
4.8.4.4.7 Ancillary images

Sometimes, an image is not critical to the content, but is nonetheless neither purely decorative nor entirely redundant with the text. In these cases, the \texttt{alt} attribute must be present, and its value should either be the empty string, or a textual representation of the information that the image conveys. If the image has a caption giving the image's title, then the \texttt{alt} attribute's value must not be empty (as that would be quite confusing for non-visual readers).

\begin{example}
Consider a news article about a political figure, in which the individual's face was shown in an image that, through a style sheet, is floated to the right. The image is not purely decorative, as it is relevant to the story. The image is not entirely redundant with the story either, as it shows what the politician looks like. Whether any alternative text need be provided is an authoring decision, in part influenced by whether the image colors the interpretation of the prose.

In this first variant, the image is shown without context, and no alternative text is provided:

\begin{verbatim}
\textless p \textless img \textit{src}="alexsalmond.jpeg" \textit{alt}="" \textgreater
Ahead of today's referendum, the First Minister of Scotland, Alex Salmond, wrote an open letter to all registered voters. In it, he admitted that all countries make mistakes.\textless /p\textgreater
\end{verbatim}

If the picture is just a face, there might be no value in describing it. It's of no interest to the reader whether the individual has red hair or blond hair, whether the individual has white skin or black skin, whether the individual has one eye or two eyes.

However, if the picture is more dynamic, for instance showing the politician as angry, or particularly happy, or devastated, some alternative text would be useful in setting the tone of the article, a tone that might otherwise be missed:

\begin{verbatim}
\textless p \textless img \textit{src}="alexsalmond.jpeg" \textit{alt}="Alex Salmond is sad." \textgreater
Ahead of today's referendum, the First Minister of Scotland, Alex Salmond, wrote an open letter to all registered voters. In it, he admitted that all countries make mistakes.\textless /p\textgreater
\end{verbatim}

\begin{verbatim}
\textless p \textless img \textit{src}="alexsalmond.jpeg" \textit{alt}="Alex Salmond is ecstatic!" \textgreater
Ahead of today's referendum, the First Minister of Scotland, Alex Salmond, wrote an open letter to all registered voters. In it, he admitted that all countries make mistakes.\textless /p\textgreater
\end{verbatim}

Whether the individual was "sad" or "ecstatic" makes a difference to how the rest of the paragraph is to be interpreted: is he likely saying that he is resigned to the populace making a bad choice in the upcoming referendum, or is he saying that the election was a mistake but the likely turnout will make it irrelevant? The interpretation varies based on the image.

\begin{example}
If the image has a caption, then including alternative text avoids leaving the non-visual user confused as to what the caption refers to.

\begin{verbatim}
\textless p\textless figcaption> Ahead of today's referendum, the First Minister of Scotland, Alex Salmond, wrote an open letter to all registered voters. In it, he admitted that all countries make mistakes.\textless /p\textless figure\textless img \textit{src}="alexsalmond.jpeg" \textit{alt}="A high forehead, cheerful disposition, and dark hair round out Alex Salmond's face." \textgreater
\textless figcaption> Alex Salmond, SNP. Photo © 2014 PolitiPhoto. \textless /figcaption\textgreater
\end{verbatim}
\end{example}
If an image is decorative but isn't especially page-specific — for example an image that forms part of a site-wide design scheme — the image should be specified in the site's CSS, not in the markup of the document.

However, a decorative image that isn't discussed by the surrounding text but still has some relevance can be included in a page using the `img` element. Such images are decorative, but still form part of the content. In these cases, the `alt` attribute must be present but its value must be the empty string.

Example

Examples where the image is purely decorative despite being relevant would include things like a photo of the Black Rock City landscape in a blog post about an event at Burning Man, or an image of a painting inspired by a poem, on a page reciting that poem. The following snippet shows an example of the latter case (only the first verse is included in this snippet):

```
<h1>The Lady of Shalott</h1>
<p><img src="shalott.jpeg" alt=""></p>
<p>On either side the river lie<br>
Long fields of barley and of rye,<br>
That clothe the wold and meet the sky;<br>
And through the field the road run by<br>
To many-tower'd Camelot;<br>
And up and down the people go,<br>
Gazing where the lilies blow<br>
Round an island there below,<br>
The island of Shalott.</p>
```

4.8.4.4.4 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, one of the images must have its `alt` attribute set as per the relevant rules that would be appropriate for the picture as a whole, and then all the remaining images must have their `alt` attribute set to the empty string.

Example

In the following example, a picture representing a company logo for XYZ Corp has been split into two pieces, the first containing the letters "XYZ" and the second with the word "Corp". The alternative text ("XYZ Corp") is all in the first image.

```
<h1><img src="logo1.png" alt="XYZ Corp"><img src="logo2.png" alt=""></h1>
```

Example

In the following example, a rating is shown as three filled stars and two empty stars. While the alternative text could have been "★★★☆☆", the author has instead decided to more helpfully give the rating in the form "3 out of 5". That is the alternative text of the first image, and the rest have blank alternative text.

```
<p>Rating: <meter max=5 value=3><img src="1" alt="3 out of 5"></img> <img src="1" alt=""> <img src="1" alt=""> <img src="0" alt=""
> <img src="0" alt=""></meter></p>
```

4.8.4.4.10 A group of images that form a single larger picture with links

Generally, image maps should be used instead of slicing an image for links.

However, if an image is indeed sliced and any of the components of the sliced picture are the sole contents of links, then one image per link must have alternative text in its `alt` attribute representing the purpose of the link.

Example

356
In some cases, the image is a critical part of the content. This could be the case, for instance, on a page that is part of a photo gallery. The image is the whole point of the page containing it.

How to provide alternative text for an image that is a key part of the content depends on the image's provenance.

The general case

When it is possible for detailed alternative text to be provided, for example if the image is part of a series of screenshots in a magazine review, or part of a comic strip, or is a photograph in a blog entry about that photograph, text that can serve as a substitute for the image must be given as the contents of the `alt` attribute.

Example

A screenshot in a gallery of screenshots for a new OS, with some alternative text:

```html
<figure>
  <img src="KDE%20Light%20desktop.png" alt="The desktop is blue, with icons along the left hand side in two columns, reading System, Home, K-Mail, etc. A window is open showing that menus wrap to a second line if they cannot fit in the window. The window has a list of icons along the top, with an address bar below it, a list of icons for tabs along the left edge, a status bar on the bottom, and two panes in the middle. The desktop has a bar at the bottom of the screen with a few buttons, a pager, a list of open applications, and a clock.">
  <figcaption>Screenshot of a KDE desktop.</figcaption>
</figure>
```

Example

A graph in a financial report:

```html
<img src="sales.gif" title="Sales graph" alt="From 1998 to 2005, sales increased by the following percentages with each year: 624%, 75%, 138%, 40%, 35%, 9%, 21%">
```

Note that "sales graph" would be inadequate alternative text for a sales graph. Text that would be a good caption is not generally suitable as replacement text.

Images that defy a complete description

In certain cases, the nature of the image might be such that providing thorough alternative text is impractical. For example, the image could be indistinct, or could be a complex fractal, or could be a detailed topographical map.

In these cases, the `alt` attribute must contain some suitable alternative text, but it may be somewhat brief.
Sometimes there simply is no text that can do justice to an image. For example, there is little that can be said to usefully describe a Rorschach inkblot test. However, a description, even if brief, is still better than nothing:

```html
<figure>
  <img src="/commons/a/a7/Rorschach1.jpg" alt="A shape with left-right symmetry with indistinct edges, with a small gap in the center, two larger gaps offset slightly from the center, with two similar gaps under them. The outline is wider in the top half than the bottom half, with the sides extending upwards higher than the center, and the center extending below the sides."/>
  <figcaption>A black outline of the first of the ten cards in the Rorschach inkblot test.</figcaption>
</figure>
```

Note that the following would be a very bad use of alternative text:

```html
!--- This example is wrong. Do not copy it. --->
<figure>
  <img src="/commons/a/a7/Rorschach1.jpg" alt="A black outline of the first of the ten cards in the Rorschach inkblot test."/>
  <figcaption>A black outline of the first of the ten cards in the Rorschach inkblot test.</figcaption>
</figure>
```

Including the caption in the alternative text like this isn't useful because it effectively duplicates the caption for users who don't have images, taunting them twice yet not helping them any more than if they had only read or heard the caption once.

Another example of an image that defies full description is a fractal, which, by definition, is infinite in detail.

The following example shows one possible way of providing alternative text for the full view of an image of the Mandelbrot set.

```html
<figure>
  <img src="ms1.jpeg" alt="The Mandelbrot set appears as a cardioid with its cusp on the real axis in the positive direction, with a smaller bulb aligned along the same center line, touching it in the negative direction, and with these two shapes being surrounded by smaller bulbs of various sizes."/>
</figure>
```

Similarly, a photograph of a person's face, for example in a biography, can be considered quite relevant and key to the content, but it can be hard to fully substitute text for:

```html
<section class="bio">
  <h1>A Biography of Isaac Asimov</h1>
  Born Isaak Yudovich Ozimov in 1920, Isaac was a prolific author.
  Later in life, he wore long white sideburns.
  Asimov was born in Russia, and moved to the US when he was three years old.
</section>
```

In such cases it is unnecessary (and indeed discouraged) to include a reference to the presence of the image itself in the alternative text, since such text would be redundant with the browser itself reporting the presence of the image. For example, if the alternative text was "A photo of Isaac Asimov", then a conforming user agent might read that out as "(Image) A photo of Isaac Asimov" rather than the more useful "(Image) Isaac Asimov had dark hair, a tall forehead, and wore glasses...".
using user-provided images where the user did not provide suitable or usable alternative text (e.g. photograph sharing sites), or because the author does not themself know what the images represent (e.g. a blind photographer sharing an image on their blog).

In such cases, the `alt` attribute may be omitted, but one of the following conditions must be met as well:

- The `img` element is in a `figure` element that contains a `figcaption` element that contains content other than inter-element whitespace, and, ignoring the `figcaption` element and its descendants, the `figure` element has no flow content descendants other than inter-element whitespace and the `img` element.

- The `title` attribute is present and has a non-empty value.

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).

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

A photo on a photo-sharing site, if the site received the image with no metadata other than the caption, could be marked up as follows:

```html
<article>
  <h1>I took a photo</h1>
  <p>I went out today and took a photo!</p>
  <figure>
    <img src="1100670787_6a7c664aef.jpg">
    <figcaption>Bubbles traveled everywhere with us.</figcaption>
  </figure>
</article>
```

It would be better, however, if a detailed description of the important parts of the image obtained from the user and included on the page.

Example

A blind user's blog in which a photo taken by the user is shown. Initially, the user might not have any idea what the photo they took shows:

```html
<article>
  <h1>I took a photo</h1>
  <p>I went out today and took a photo!</p>
  <figure>
    <img src="photo2.jpeg"
    alt="The photograph shows my squirrel feeder hanging from the edge of my roof. It is half full, but there are no squirrels around. In the background, out-of-focus trees fill the shot. The feeder is made of wood with a metal grate, and it contains peanuts. The edge of the roof is wooden too, and is painted white with light blue streaks."/>
    <figcaption>A photograph taken blindly from my front porch.</figcaption>
  </figure>
</article>
```

Eventually though, the user might obtain a description of the image from their friends and could then include alternative text:
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, then the `alt` attribute must be the empty string. In such cases, the `width` and `height` attributes should both be set to zero.

This section does not apply to documents that are publicly accessible, or whose target audience is not necessarily personally known to the author, such as documents on a web site, emails sent to public mailing lists, or software documentation.

When an image is included in a private communication (such as an HTML email) aimed at a specific person who is known to be able to view images, the `alt` attribute may be omitted. However, even in such cases authors are strongly urged to include alternative text (as appropriate according to the kind of image involved, as described in the above entries), so that the email is still usable should the user use a mail client that does not support images, or should the document be forwarded on to other users whose abilities might not allow them to view the image.

Example

Sometimes the entire point of the image is that a textual description is not available, and the user is to provide the description. For instance, the point of a CAPTCHA image is to see if the user can literally read the graphic. Here is one way to mark up a CAPTCHA (note the `title` attribute):

```html
<p><label>What does this image say?</label><img src="captcha.cgi?id=8934" title="CAPTCHA"><input type="text" name="captcha"></p>
```

(If you cannot see the image, you can use an `<a href="?audio">audio</a>` test instead.)

Another example would be 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:

```html
<table>
<thead>
<tr><th>Image</th><th>Description</th></tr>
</thead>
<tbody>
<tr><td><img src="2421.png" title="Image 640 by 100, filename 'banner.gif'"> <input name="alt2421"></td></tr>
<tr><td><img src="2422.png" title="Image 200 by 480, filename 'ad3.gif'"> <input name="alt2422"></td></tr>
</tbody>
</table>
```

Notice that even in this example, as much useful information as possible is still included in the `title` attribute.

Note

Since some users cannot use images at all (e.g. because they have a very slow connection, or because they are using a text-only browser, or because they are listening to the page being read out by a hands-free automobile voice web browser, or simply because they are blind), the `alt` attribute is only allowed to be omitted rather than being provided with replacement text when no alternative text is available and none can be made available, as in the above examples. Lack of effort from the part of the author is not an acceptable reason for omitting the `alt` attribute.

4.8.4.4.12 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, then the `alt` attribute must be the empty string.

In such cases, the `width` and `height` attributes should both be set to zero.

4.8.4.4.13 An image in an email or private document intended for a specific person who is known to be able to view images

This section does not apply to documents that are publicly accessible, or whose target audience is not necessarily personally known to the author, such as documents on a web site, emails sent to public mailing lists, or software documentation.

When an image is included in a private communication (such as an HTML email) aimed at a specific person who is known to be able to view images, the `alt` attribute may be omitted. However, even in such cases authors are strongly urged to include alternative text (as appropriate according to the kind of image involved, as described in the above entries), so that the email is still usable should the user use a mail client that does not support images, or should the document be forwarded on to other users whose abilities might not allow them to view the image.
include easily seeing images.

### 4.8.4.4.14 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, or the `title` attribute, to provide the image's caption.

As a last resort, implementers 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 `img` elements for which they have been unable to obtain alternative text 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 alternative text, because state-of-the-art automated conformance checkers cannot distinguish phony alternative text from correct alternative text.

**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.

### 4.8.4.4.15 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 `img` element is in a `figure` element that satisfies the conditions described above.
- The `img` element has a `title` attribute with a value that is not the empty string (also as described above).
- The conformance checker has been configured to assume that the document is an email or document intended for a specific person who is known to be able to view images.
- The `img` 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.)
4.8.5 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:
Nothing.

Tag omission in text/html:
Neither tag is omissible.

Content attributes:
- src — Address of the resource
- srcdoc — A document to render in the iframe's nested browsing context
- name — Name of nested browsing context
- sandbox — Security rules for nested content
- allow — Permissions policy to be applied to the iframe's contents
- allowfullscreen — Whether to allow the iframe's contents to use requestFullscreen()
- width — Horizontal dimension
- height — Vertical dimension
- referrerPolicy — Referrer policy for fetches initiated by the element
- loading — Used when determining loading deferral

Accessibility considerations:
For authors.
For implementers.

DOM interface:

```
IDL [Exposed=Window]
interface HTMLIFrameElement : HTMLElement {
  [HTMLConstructor] constructor();

  [CEReactions] attribute USVString src;
  [CEReactions] attribute DOMString srcdoc;
  [CEReactions] attribute DOMString name;
  [SameObject, PutForwards=value] readonly attribute DOMTokenList sandbox;
  [CEReactions] attribute DOMString allow;
  [CEReactions] attribute DOMString allowFullScreen;
  [CEReactions] attribute DOMString width;
  [CEReactions] attribute DOMString height;
  [CEReactions] attribute DOMString referrerPolicy;
  [CEReactions] attribute DOMString loading;
  readonly attribute Document? contentDocument;
  readonly attribute WindowProxy? contentWindow;
  Document? getSVGDocument();

  // also has obsolete members
};
```

The iframe element represents its nested browsing context.

The src attribute gives the URL of a page that the element's nested browsing context is to contain. The attribute, if present, must be a valid non-empty URL potentially surrounded by spaces. If the itemprop attribute is specified on an iframe element, then the src attribute must also be specified.
The `srcdoc` attribute gives the content of the page that the element's 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 ASCII whitespace.
2. Optionally, a `DOCTYPE`.
3. Any number of comments and ASCII whitespace.
4. The document element, in the form of an `html` element.
5. Any number of comments and ASCII whitespace.

The above requirements apply in XML documents as well.

**Note**

The above requirements apply in XML documents as well.

**Example**

Here a blog uses the `srcdoc` attribute in conjunction with the `sandbox` attribute 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:

```html
<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>
    Written by <a href="/users/cap">cap</a>, 1 hour ago.
  </footer>
</article>

<article>
  <footer>Thirteen minutes ago, <a href="/users/ch">ch</a> wrote: </footer>
  <iframe sandbox srcdoc="<p>did you get a cover picture yet?</p>"></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;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. you should get earl&amp;amp;me on the next cover.</p>"></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+0026 AMPERSAND (&) and U+0022 QUOTATION MARK ("•") characters, and to specify the
When an iframe element is inserted into a document whose browsing context is non-null, the user agent must run these steps:

1. Create a new nested browsing context for element.
2. Process the iframe attributes for element, with initialInsertion set to true.

When an iframe element is removed from a document, the user agent must discard the element's nested browsing context, if it is not null, and then set the element's nested browsing context to null.

Whenever an iframe element with a non-null 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 non-null nested browsing context but with no srcdoc attribute specified has its src attribute specified has its src attribute set, changed, or removed, the user agent must process the iframe attributes.

To process the iframe attributes for an element with an optional boolean initialInsertion (default false):

1. If element's srcdoc attribute is specified, then:
   1. Set element's current navigation was lazy loaded boolean to false.
   2. If the will lazy load element steps given element return true, then:
      1. Set element's lazy load resumption steps to the rest of this algorithm starting with the step labeled navigate to the srcdoc resource.
      2. Set element's current navigation was lazy loaded boolean to true.
      3. Start intersection-observing a lazy loading element for element.
      4. Return.
   3. Navigate to the srcdoc resource: navigate an iframe or frame given element and a new response whose url list consists of about:srcdoc, header list consists of `Content-Type`/`text/html`, body is the value of element's srcdoc attribute, and CSP list is a clone of element's node document's CSP list.

   The resulting Document must be considered an iframe srcdoc document.

2. Otherwise, if element has a src attribute specified, or initialInsertion is false, then run the shared attribute processing steps for iframe and frame elements given element.

The shared attribute processing steps for iframe and frame elements, given an element, are:

1. If element has no src attribute specified, or its value is the empty string, let url be the URL "about:blank".
Otherwise, parse the value of element's `src` attribute, relative to element's node document. If that is not successful, then let `url` be the URL "about:blank". Otherwise, let `url` be the resulting URL record.

2. If there exists an ancestor browsing context of element's nested browsing context whose active document's URL, ignoring fragments, is equal to `url`, then return.

3. Let `resource` be a new request whose `url` is `url` and whose referrer policy is the current state of element's `referrerpolicy` content attribute.

4. If `element` is an iframe element, then set `element`'s current navigation was lazy loaded boolean to false.

5. If `element` is an iframe element, and the will lazy load element steps given `element` return true, then:
   1. Set `element`'s lazy load resumption steps to the rest of this algorithm starting with the step labeled navigate to the resource.
   2. Set `element`'s current navigation was lazy loaded boolean to true.
   3. Start intersection-observing a lazy loading element for `element`.
   4. Return.

6. Navigate to the resource: navigate an iframe or frame given `element` and `resource`.

To navigate an iframe or frame given an element `element` and a resource `resource`:

1. Let `historyHandling` be "default".

2. If `element`'s nested browsing context's session history contains only one Document, and that was the about:blank Document created when `element`'s nested browsing context was created, then set `historyHandling` to "replace".

3. If `element`'s nested browsing context's active document is not completely loaded, then set `historyHandling` to "replace".

4. Navigate `element`'s nested browsing context to `resource`, with `historyHandling` set to `historyHandling` and the source browsing context set to `element`'s node document's browsing context.

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.

To run the iframe load event steps, given an iframe element `element`:

1. Assert: `element`'s nested browsing context is not null.

2. Let `childDocument` be the active document of `element`'s nested browsing context.

3. If `childDocument` has its mute iframe load flag set, then return.


5. Fire an event named `load` at `element`.


⚠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.

If an element type potentially delays the load event, then for each element `element` of that type, the user agent must delay the load event of element's node document if element's nested browsing context is non-null and any of the following are true:
• element's nested browsing context's active document is not ready for post-load tasks.

• Anything is delaying the load event of element's nested browsing context's active document.

• element's nested browsing context is in the delaying load events mode.

Note

If, during the handling of the load event, element's nested browsing context is again navigated, that will further delay the load event.

Each iframe element has an associated current navigation was lazy loaded boolean, initially false. It is set and unset in the process the iframe attributes algorithm.

An iframe element whose current navigation was lazy loaded boolean is false potentially delays 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 parsed, the browsing context will remain at the initial about:blank page.

Note

If the user navigates away from this page, the iframe's nested browsing context's 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 element's nested browsing context if present when that is created.

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-modals, allow-orientation-lock, allow-pointer-lock, allow-popups, allow-popups-to-escape-sandbox, allow-presentation, allow-same-origin, allow-scripts, allow-top-navigation, allow-top-navigation-by-user-activation, and allow-downloads.

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; the allow-top-navigation-by-user-activation keyword behaves similarly but allows such navigation only when the browsing context's active window has transient activation, and the allow-forms, allow-modals, allow-orientation-lock, allow-pointer-lock, allow-popups, allow-presentation, allow-scripts, and allow-popups-to-escape-sandbox keywords re-enable forms, modal dialogs, screen orientation lock, the pointer lock API, popups, the presentation API, scripts, and the creation of unsandboxed auxiliary browsing contexts, respectively. [POINTERLOCK] [SCREENORIENTATION] [PRESENTATION]

The allow-top-navigation and allow-top-navigation-by-user-activation keywords must not both be specified, as doing so is redundant; only allow-top-navigation will have an effect in such non-conformant markup.

⚠ 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 element 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 and the iframe sandboxing flag set as the output.

When an iframe element's sandbox attribute is removed while it has a non-null nested browsing context, the user agent must empty the iframe element's iframe sandboxing flag set.

Example

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).

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

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.

Example

Suppose a file A contained the following fragment:

Suppose that file B contained an iframe also:

Further, suppose that file C contained a link:

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 allow attribute, when specified, determines the container policy that will be used when the permissions policy for a Document in the iframe's nested browsing context is initialized. Its value must be a serialized permissions policy.

**Example**

In this example, an iframe is used to embed a map from an online navigation service. The allow attribute is used to enable the Geolocation API within the nested context.

```html
<iframe src="https://maps.example.com/" allow="geolocation"></iframe>
```

The allowfullscreen attribute is a boolean attribute. When specified, it indicates that Document objects in the iframe element's nested browsing context will be initialized with a permissions policy which allows the "fullscreen" feature to be used from any origin. This is enforced by the process permissions policy attributes algorithm.

**Example**

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.

```html
<article>
<header>
<p><img src="/usericons/1627591962735"> <b>Fred Flintstone</b></p>
<p><a href="/posts/3095182851" rel=bookmark>12:44</a> — <a href="#acl-3095182851">Private Post</a></p>
</header>
<p>Check out my new ride!</p>
</article>
```

Neither allow nor allowfullscreen can grant access to a feature in an iframe element's nested browsing context if the element's node document is not already allowed to use that feature.

To determine whether a Document object document is allowed to use the policy-controlled-feature feature, run these steps:

1. If document's browsing context is null, then return false.
2. If document's browsing context's active document is not document, then return false.
3. If the result of running is feature enabled in document for origin on feature, document, and document's origin is "Enabled", then return true.
4. Return false.

⚠️ Warning! Because they only influence the permissions policy of the nested browsing context's active document, the allow and allowfullscreen attributes only take effect when the nested browsing context of the iframe is navigated. Adding or removing them has no effect on an already-loaded document.
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 new nested browsing context`, regardless of whether the specified initial contents are successfully used.

The `referrerPolicy` attribute is a `referrer policy attribute`. Its purpose is to set the `referrer policy` used when `processing the iframe attributes`. [REFERRERPOLICY]

The `loading` attribute is a `lazy loading attribute`. Its purpose is to indicate the policy for loading `iframe` elements that are outside the viewport.

When the `loading` attribute's state is changed to the `Eager` state, the user agent must run these steps:

1. Let `resumptionSteps` be the `iframe` element's `lazy load resumption steps`.
2. If `resumptionSteps` is null, then return.
3. Set the `iframe`'s `lazy load resumption steps` to null.
4. Invoke `resumptionSteps`.

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.)

**Note**
The HTML parser treats markup inside `iframe` elements as text.

The IDL attributes `src`, `srcdoc`, `name`, `sandbox`, and `allow` 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 `referrerPolicy` IDL attribute must reflect the `referrerpolicy` content attribute, limited to only known values.

The `loading` IDL attribute must reflect the `loading` content attribute, limited to only known values.

The `contentDocument` IDL attribute, on getting, must return the `iframe` element's `content document`, if its `nested browsing context` is non-null, or null otherwise.

**Example**
Here is an example of a page using an `iframe` to include advertising from an advertising broker:

```html
<iframe src="https://ads.example.com/?customerid=923513721&format=banner" width="468" height="60"></iframe>
```

4.8.6 The `embed` element

**Categories**:
- Flow content
- Phrasing content
- Embedded content
- Interactive content.

---

The `HTML parser` treats markup inside `iframe` elements as text.
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).

Accessibility considerations:
For authors.
For implementers.

DOM interface:

```
[Exposed=Window]
interface HTMLEmbedElement : HTMLElement {
    [HTMLConstructor] constructor();

    [CEReactions] attribute USVString src;
    [CEReactions] attribute DOMString type;
    [CEReactions] attribute DOMString width;
    [CEReactions] attribute DOMString height;
    Document? getSVGDocument();

    // also has obsolete members
}
```

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 URL of the resource being embedded. The attribute, if present, must contain a valid non-empty URL potentially surrounded by spaces.

⚠️ Warning!
Authors should avoid referencing untrusted resources, as such a resource can be used to instantiate plugins or run scripts, even if the author has used features such as the Flash "allowScriptAccess" parameter.

If the itemprop attribute is specified on an embed element, then the src attribute must also be specified.

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 string. If both the type and 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** 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 an element task** on the **embed task source** given the element to run the **embed element setup steps** for that element.

**The embed element setup steps** for a given **embed** element `element` are as follows:

1. If another **task** has since been queued to run the **embed element setup steps** for `element`, then return.
2. If `element` has a `src` attribute set, then:
   1. Let `url` be the result of **parsing** the value of `element`'s `src` attribute, relative to `element`'s **node document**.
   2. If `url` is failure, then return.
   3. Let `request` be a new **request** whose `url` is `url`, `client` is `element`'s **node document**'s relevant settings object, `destination` is "embed", `credentials mode` is "include", `mode` is "navigate", and whose **use-URL-credentials flag** is set.
   4. Fetch `request`.

   Fetching the resource must **delay the load event** of `element`'s **node document**.

   To **process response** for the **response** `response`:
   1. If another **task** has since been queued to run the **embed element setup steps** for `element`, then return.
   2. If `response` is a **network error**, then **fire an event** named `Load` at `element`, and return.
   3. Let `type` be the result of determining the **type of content** given `element` and `response`.
   4. Switch on `type`:
      - `null`
      1. **Display no plugin** for `element`.
      - `image/svg+xml`
      1. If `element`'s **nested browsing context** is null, then create a new **nested browsing context** for `element`.
      2. **Navigate** `element`'s **nested browsing context** to `response`, with `historyHandling` set to "replace" and the **source browsing context** set to `element`'s **node document**'s **browsing context**.

      **Note**

      *element*'s `src` attribute does not get updated if the browsing context gets further navigated to other locations.*

      3. `element` now **represents** its **nested browsing context**.
      - `Otherwise`
      1. **Display a plugin** for `element`, given `type` and `response`.

3. Otherwise:
   1. Let `type` be the value of `element`'s `type` attribute.
   2. If `type` is a type that a **plugin** supports, then **display a plugin** for `element` given `type`.

371
3. Otherwise, display no plugin for element.

To determine the type of the content given an embed element element and a response response, run the following steps:

1. If element has a type attribute, and that attribute's value is a type that a plugin supports, then return the value of the type attribute.
2. If the path component of response's url matches a pattern that a plugin supports, then return the type that that plugin can handle.

Example

For example, a plugin might say that it can handle URLs with path components that end with the four character string ".swf".

3. If response has explicit Content-Type metadata, and that value is a type that a plugin supports, then return that value.
4. Return null.

To display a plugin for an embed element element, given a string type and optionally a response response:

1. If element's nested browsing context is not null, then:
   1. Discard element's nested browsing context.
   2. Set element's nested browsing context to null.
2. Find and instantiate an appropriate plugin based on type, replacing any previously-instantiated plugin for element. If response was given, forward it to the plugin.
3. element now represents this plugin instance.
4. Once the plugin, and response if given, are completely loaded, queue an element task on the DOM manipulation task source give element to fire an event named load at element.

To display no plugin for an embed element element:

1. If element's nested browsing context is not null, then:
   1. Discard element's nested browsing context.
   2. Set element's nested browsing context to null.
2. Display an indication that no plugin could be found for element, replacing any previously-instantiated plugin for element.
3. element now represents nothing.

Note

The embed element has no fallback content; its descendants are ignored.

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, 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
The `embed` element potentially delays the load event.

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 ASCII upper alphas. 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 [HTML EmbedElement](https://developer.mozilla.org/en-US/docs/Web/API/HTMLEmbedElement) object representing the element must expose the scriptable interface of the plugin instantiated for the `embed` element, if any.

The `embed` element supports dimension attributes.

The IDL attributes `src` and `type` each must reflect the respective content attributes of the same name.

**Example**

Here's a way to embed a resource that requires a proprietary plugin, like Flash:

```html
<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:

```html
<embed src="catgame.swf" quality="high">
```

This would be equivalent to the following, when using an `object` element instead:

```html
<object data="catgame.swf">
  <param name="quality" value="high">
</object>
```

### 4.8.7 The `object` element

**Categories:**
- Flow content
- Phrasing content
- Embedded content

If the element has a `usemap` attribute: Interactive content.

Listed, 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
  - `name` — Name of nested `browsing context`.
  - `usemap` — Name of `image map` to use
  - `form` — Associates the element with a `form` element
  - `width` — Horizontal dimension
  - `height` — Vertical dimension

Accessibility considerations:
  - For authors.
  - For implementers.

DOM interface:
```idl
[Exposed=Window]
interface HTMLObjectElement : HTMLElement {
  [HTMLConstructor] constructor();

  [CEReactions] attribute USVString data;
  [CEReactions] attribute DOMString type;
  [CEReactions] attribute DOMString name;
  [CEReactions] attribute DOMString useMap;
  readonly attribute HTMLFormElement? form;
  [CEReactions] attribute DOMString width;
  [CEReactions] attribute DOMString height;
  readonly attribute Document? contentDocument;
  readonly attribute WindowProxy? contentWindow;
  Document? getSVGDocument();

  readonly attribute boolean willValidate;
  readonly attribute ValidityState validity;
  readonly attribute DOMString validationMessage;
  boolean checkValidity();
  boolean reportValidity();
  undefined setCustomValidity(DOMString error);

  // also has obsolete members
};
```

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 `child browsing context`, or as an external resource to be processed by a `plugin`.

The `data` attribute, if present, specifies the URL of the resource. If present, the attribute must be a valid non-empty URL potentially surrounded by spaces.

**Warning!**
Authors should avoid referencing untrusted resources, as such a resource can be used to instantiate plugins or run 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 string**.

At least one of either the **data** attribute or the **type** attribute must be present.

If the **itemprop** attribute is specified on an **object** element, then the **data** attribute must also be specified.

The **name** attribute, if present, must be a **valid browsing context name**. The given value is used to name the element's **nested browsing context**, if applicable, and if present when the element's **nested browsing context** is created.

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**, if any, or if the element is not on the **传感度** **或** **XML parser**, and it is either inserted into a **document** or removed from a **document**, if any,
- 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 an element task**, on the **DOM manipulation task source**, given the **object** element 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 whose **browsing context** is non-null, 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 a URL** given the **data** attribute, relative to the element's **node document**.
   3. If that failed, **fire an 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 record**, **client** is the element's **node document**'s **relevant settings object**, **destination** is "object", **credentials mode** is "include", **mode** is "navigate", and whose **use-URL-credentials flag** is set.
   5. **Fetch** **request**.

375
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 an 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 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.

   3. 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.

   4. 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.

      5. 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 that plugin can handle.
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 the `object` element's `nested browsing context` is non-null, then it must be discarded and then set to null.
  
  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/"**
  
  If the `object` element's `nested browsing context` is null, then create a new `nested browsing context` for the element.
  
  If the URL of the given resource is not `about:blank`, then navigate the element's `nested browsing context` to that resource, with `historyHandling` set to "replace" and the source `browsing context` set to the `object` element's `node document`'s `browsing context`. (The `data` attribute of the `object` element doesn't get updated if the browsing context gets further navigated to other locations.)
  
  The `object` element represents its `nested browsing context`.

- **If the resource type starts with "image/", and support for images has not been disabled**
  
  If the `object` element's `nested browsing context` is non-null, then it must be discarded and then set to null.
  
  Apply the image sniffing rules to determine the type of the image.
  
  The `object` element represents the specified image.
  
  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. If the `object` element does not represent its `nested browsing context`, then once the resource is completely loaded, queue an element task on the DOM manipulation task source given the `object` element to fire an event named `load` at the element.

  **Note**
  
  If the element does represent its `nested browsing context`, then an analogous task will be queued when the created `Document` is completely finished loading.

12. Return.

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 return; once the plugin is completely loaded, queue an element task on the DOM manipulation task source, given the object element to fire an 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. If the element’s nested browsing context is non-null, then it must be discarded and then set to null.

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.

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.

The object element potentially delays the load event.

The usemap attribute, if present while the object element represents an image, can indicate that the object has an associated image map. The attribute must be ignored if the object element doesn’t represent an image.

The form attribute is used to explicitly associate the object element with its form owner.

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 useMap IDL attribute must reflect the usemap content attribute.

The contentDocument IDL attribute, on getting, must return the object element’s content document.

The contentWindow IDL attribute must return the WindowProxy object of the object element’s nested browsing context, if its nested browsing context is non-null; 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.

Example

In this example, an HTML page is embedded in another using the object element.

```html
<figure>
<object data="clock.html"></object>
<figcaption>My HTML Clock</figcaption>
</figure>
```

Example

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.

```html
<p>Look at my video:
<object type="application/x-shockwave-flash">
<param name="movie" value="https://video.example.com/library/watch.swf">
</object>
</p>
```
4.8.8 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

**Accessibility considerations**: For authors. For implementers.

**DOM interface**:

```javascript
[Exposed=Window]
interface HTMLParamElement : HTMLElement {
  [HTMLConstructor] constructor();

  [CFReactions] attribute DOMString name;
  [CFReactions] attribute DOMString value;

  // also has obsolete members
};
```

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.

```html
<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>
```
The following example shows how the `param` element can be used to pass a parameter to a plugin, in this case the O3D plugin.

```html
<!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">
  <img src="o3d-teapot.png" title="3D Utah Teapot illustration rendered using O3D."
   alt="When O3D renders the Utah Teapot, it appears as a squat teapot with a shiny metallic finish on which the surroundings are reflected, with a faint shadow caused by the lighting.">
</object>
<p>To see the teapot actually rendered by O3D on your computer, please download and install the <a href="http://code.google.com/apis/o3d/docs/gettingstarted.html#install">O3D plugin</a>.</p>
<script src="o3d-teapot.js"></script>
</p>
</body>
</html>
```

### 4.8.9 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
  - `playsinline` — Encourage the user agent to display video content within the element's playback area
  - `loop` — Whether to loop the media resource
  - `muted` — Whether to mute the media resource by default
  - `controls` — Show user agent controls
  - `width` — Horizontal dimension
Accessibility considerations:

For authors.
For implementers.

DOM interface:

```idl
[Exposed=Window]
interface HTMLVideoElement : HTMLMediaElement {
    [HTMLConstructor] constructor();

    [CFReactions] attribute unsigned long width;
    [CFReactions] attribute unsigned long height;
    readonly attribute unsigned long videoWidth;
    readonly attribute unsigned long videoHeight;
    [CFReactions] attribute USVString poster;
    [CFReactions] attribute boolean playsInline;
}
```

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.

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 embedded in the video stream. Audio descriptions can be 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.

The `video` element is a `media element` whose `media data` is ostensibly video data, possibly with associated audio data.

The `src`, `crossorigin`, `preload`, `autoplay`, `loop`, `muted`, and `controls` attributes are the attributes common to all media elements.

The `poster` attribute gives the URL 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`; return.
3. Parse the `poster` attribute's value relative to the element's `node document`. If this fails, then there is no `poster frame`; return.
4. Let `request` be a new request whose `url` is the `resulting URL record`, `client` is the element's `node document`'s relevant settings object, `destination` is "image", `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`.
The `playsinline` attribute is a boolean attribute. If present, it serves as a hint to the user agent that the video ought to be displayed "inline" in the document by default, constrained to the element's playback area, instead of being displayed fullscreen or in an independent resizable window.

The absence of the `playsinline` attributes does not imply that the video will display fullscreen by default. Indeed, most user agents have chosen to play all videos inline by default, and in such user agents the `playsinline` attribute has no effect.

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 `HOLDING` 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.

- 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.

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.
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 media element task given the media element to fire an 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.

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.

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, such as fullscreen or in an independent resizable window. User agents may even trigger such a viewing mode by default upon playing a video, although they should not do so when the playsinline attribute is specified. As with 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 viewing mode, 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.

The playsInline IDL attribute must reflect the playsinline content attribute.

Example

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>

4.8.10 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

Accessibility considerations:
For authors.
For implementers.

DOM interface:

```idl
[Exposed=Window,
 LegacyFactoryFunction=Audio(optional DOMString src)]
interface HTMLAudioElement : HTMLMediaElement {
[HTMLConstructor] constructor();
};
```

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.

The `audio` element is a `media element` whose `media data` is ostensibly audio data.

The `src`, `crossorigin`, `preload`, `autoplay`, `loop`, `muted`, and `controls` attributes are the attributes common to all media elements.

#### For web developers (non-normative)

```javascript
audio = new Audio([url])
```

Returns a new `audio` element, with the `src` attribute set to the value passed in the argument, if applicable.

A legacy factory function is provided for creating `HTMLAudioElement` objects (in addition to the factory methods from DOM such as `createElement()`: `Audio(src)`). When invoked, the legacy factory function must perform the following steps:

1. Let `document` be the `current global object`'s associated `Document`.
2. Let `audio` be the result of creating an element given `document`, `audio`, and the `HTML namespace`.
3. Set an attribute value for `audio` using `"preload"` and `"autoplay"`.
4. If `src` is given, then set an attribute value for `audio` using `"src"` and `src`. (This will cause the user agent to invoke the object's resource selection algorithm before returning.)
5. Return `audio`.

#### 4.8.11 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

Accessibility considerations:
- For authors.
- For implementers.

DOM interface:

```idl
[Exposed=Window]
interface HTMLTrackElement : HTMLElement {
    [HTMLConstructor] constructor();

    [CEReactions] attribute DOMString kind;
    [CEReactions] attribute USVString src;
    [CEReactions] attribute DOMString srclang;
    [CEReactions] attribute DOMString label;
    [CEReactions] 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 timed text tracks 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.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>State</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subtitles</td>
<td>Subtitles</td>
<td>Transcription or translation of the dialogue, 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.</td>
</tr>
<tr>
<td>captions</td>
<td>Captions</td>
<td>Transcription or translation of the dialogue, 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.</td>
</tr>
<tr>
<td>descriptions</td>
<td>Descriptions</td>
<td>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.</td>
</tr>
<tr>
<td>chapters</td>
<td>Chapters</td>
<td>Tracks intended for use from script. Not displayed by the user agent.</td>
</tr>
<tr>
<td>metadata</td>
<td>Metadata</td>
<td></td>
</tr>
</tbody>
</table>

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 URL 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 **chapters metadata** or...
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 description 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 metadata 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.

For web developers (non-normative)

**track**.readyState

`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.

- 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.
The **track** IDL attribute must, on getting, return the **track** element's **text track**'s corresponding **TextTrack** object.

The **src**, **srclang**, 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**

This video has subtitles in several languages:

```html
<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.)

---

### 4.8.12 Media elements

**HTMLMediaElement** objects (**audio** and **video**, in this specification) are simply known as **media elements**.

```idl
enum CanPlayTypeResult { "" /* empty string */, "maybe", "probably" };
typedef (MediaStream or MediaSource or Blob) MediaProvider;

[Exposed=Window]
interface HTMLMediaElement : HTMLElement {

  // error state
  readonly attribute MediaError? error;

  // network state
  [CEReactions] attribute USVString src;
  attribute MediaProvider? srcObject;
  readonly attribute USVString currentSrc;
  [CEReactions] 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;
  [CEReactions] attribute DOMString preload;
  readonly attribute TimeRanges buffered;
  undefined 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;
```

---

**ERROR** (numeric value 3)

The **text track failed to load** state.

---
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.

Each media element has a unique media element event task source.

To queue a media element task with a media element element and a series of steps steps, queue an element task on the media element's media element event task source, given element and steps.

4.8.12.1 Error codes

For web developers (non-normative)

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.

**IDL**

```idl
[Exposed=Window]
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;
    readonly attribute DOMString message;
};
```

For web developers (non-normative)

- `media . error . code` Returns the current error's error code, from the list below.
- `media . error . message` Returns a specific informative diagnostic message about the error condition encountered. The message and message format are not generally uniform across different user agents. If no such message is available, then the empty string is returned.

Every MediaError object has a message, which is a string, and a code, which is 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.

To create a MediaError, given an error code which is one of the above values, return a new MediaError object whose code is the given error code and whose message is a string containing any details the user agent is able to supply about the cause of the error condition, or the empty string if the user agent is unable to supply such details. This message string must not contain only the information already available via the supplied error code; for example, it must not simply be a translation of the code into a string format. If no additional information is available beyond that provided by the error code, the message must be set to the empty string.

The code attribute of a MediaError object must return this MediaError object's code.

The message attribute of a MediaError object must return this MediaError object's message.

### 4.8.12.2 Location of the media resource

The `src` content attribute on media elements gives the URL of the media resource (video, audio) to show. The attribute, if present, must contain a valid non-empty URL potentially surrounded by spaces.

If the `itemprop` attribute is specified on the media element, then the `src` attribute must also be specified.

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, limited to only known values.

A `media provider object` is an object that can represent a `media resource`, separate from a `URL`, `MediaStream` objects, `MediaSource` objects, and `Blob` 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`.

The `currentSrc` IDL attribute must initially be set to 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`.

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.)

Types are usually somewhat incomplete descriptions; for example "video/mp4" 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.

A type that the user agent knows it cannot render 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`.

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.

For web developers (non-normative)

`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`.

For web developers (non-normative)

`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\(^{391}\), or is the type "application/octet-stream"; it must return "probably" if the user agent is confident that the type represents a media resource\(^{389}\) that it can render if used in with this `audio`\(^{384}\) or `video`\(^{380}\) element; and it must return "maybe" otherwise. Implementors are encouraged to return "maybe"\(^{392}\) unless the type can be confidently established as being supported or not. Generally, a user agent should never return "probably"\(^{392}\) for a type that allows the `codecs` parameter if that parameter is not present.

**Example**

This script tests to see if the user agent supports a (fictional) new format to dynamically decide whether to use a `video`\(^{388}\) element or a plugin:

```javascript
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);
}
```

**Note**

The `type`\(^{318}\) attribute of the `source`\(^{318}\) element allows the user agent to avoid downloading resources that use formats it cannot render.

### 4.8.12.4 Network states\(^{39}\)

For web developers (non-normative)

```html
media . networkState
```

Returns the current state of network activity for the element, from the codes in the list below.

As `media`\(^{388}\) 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`\(^{394}\) is active and has selected a `resource`\(^{389}\), 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.
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.

4.8.12.5 Loading the media resource

```javascript
media.load()
```

Causes the element to reset and start selecting and loading a new media resource from scratch.

All media elements have a can autoplay 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. Let pending tasks be a list of all tasks from the media element's media element event task source in one of the task queues.
3. For each task in pending tasks that would resolve pending play promises or reject pending play promises, immediately resolve or reject those promises in the order the corresponding tasks were queued.
4. Remove each task in pending tasks from its task queue.
5. If the media element's networkState is set to NETWORK_LOADING or NETWORK_IDLE, queue a media element task given the media element to fire an event named abort at the media element.
6. If the media element's networkState is not set to NETWORK_EMPTY, then:
   1. Queue a media element task given the media element to fire an 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:
   1. Set the paused attribute to true.
   2. Take pending play promises and reject pending play promises with the result and an "AbortError" DOMException.
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 media element task given the media element to fire an event named timeupdate at the media element.
9. Set the timeline offset\(^4\) to Not-a-Number (NaN).

10. Update the duration\(^4\) attribute to Not-a-Number (NaN).

---

**Note**

The user agent will not\(^4\) fire a durationchange\(^4\) event for this particular change of the duration.

---

7. Set the playbackRate\(^4\) attribute to the value of the defaultPlaybackRate\(^4\) attribute.

8. Set the error\(^4\) attribute to null and the can autoplay flag\(^3\) to true.

9. Invoke the media element\(^3\)'s resource selection algorithm\(^3\).

---

**Note**

Playback of any previously playing media resource\(^3\) for this element stops.

---

The **resource selection algorithm** for a media element\(^3\) is as follows. This algorithm is always invoked as part of a task\(^3\), but one of the first steps in the algorithm is to return and continue running the remaining steps in parallel\(^4\). In addition, this algorithm interacts closely with the event loop\(^6\) mechanism; in particular, it has synchronous sections\(^6\) (which are triggered as part of the event loop\(^6\) algorithm). Steps in such sections are marked with ⌛.

1. Set the element’s networkState\(^2\) attribute to the NETWORK_NO_SOURCE\(^3\) value.

2. Set the element’s show poster flag\(^4\) to true.

3. Set the media element\(^3\)'s delaying-the-load-event flag\(^3\) to true (this delays the load event\(^\)).

4. Await a stable state\(^3\), allowing the task\(^3\) that invoked this algorithm to continue. The synchronous section\(^3\) consists of all the remaining steps of this algorithm until the algorithm says the synchronous section\(^3\) has ended. (Steps in synchronous sections\(^3\) are marked with ⌛.)

5. ⌛ If the media element\(^3\)'s blocked-on-parser\(^2\) flag is false, then populate the list of pending text tracks\(^3\).

6. ⌛ If the media element\(^3\) has an assigned media provider object\(^3\), then let mode be object.

7. Otherwise, if the media element\(^3\) has no assigned media provider object\(^3\) but has a src\(^3\) attribute, then let mode be attribute.

8. Otherwise, if the media element\(^3\) does not have an assigned media provider object\(^3\) and does not have a src\(^3\) attribute, but does have a source\(^3\) element child, then let mode be children and let candidate be the first such source\(^3\) element child in tree order.

9. Otherwise the media element\(^3\) has no assigned media provider object\(^3\) and has neither a src\(^3\) attribute nor a source\(^3\) element child:
   1. ⌛ Set the networkState\(^3\) to NETWORK_EMPTY\(^3\).
   2. ⌛ Set the element’s delaying-the-load-event flag\(^3\) to false. This stops delaying the load event\(^\).
   3. End the synchronous section\(^3\) and return.

10. ⌛ Set the media element\(^3\)'s networkState\(^3\) to NETWORK_LOADING\(^3\).

11. ⌛ Queue a media element task\(^3\) given the media element\(^3\) to fire an event named loadstart\(^4\) at the media element\(^3\).

12. Run the appropriate steps from the following list:

   1. If mode is object
      1. ⌛ Set the currentSrc\(^3\) attribute to the empty string.
      2. End the synchronous section\(^3\), continuing the remaining steps in parallel\(^4\).
      3. Run the resource fetch algorithm\(^3\) with the assigned media provider object\(^3\). If that algorithm returns without aborting this one, then the load failed.
      4. Failed with media provider: Reaching this step indicates that the media resource failed to load. Take pending
play promises\textsuperscript{411} and queue a media element task\textsuperscript{409} given the media element\textsuperscript{388} to run the dedicated media source failure steps\textsuperscript{396} with the result.

5. Wait for the task\textsuperscript{403} queued by the previous step to have executed.

6. Return. The element won’t attempt to load another resource until this algorithm is triggered again.

\textbf{If mode is attribute}

1. \textit{\textbf{If} the \texttt{src}\textsuperscript{396} attribute’s value is the empty string, then end the synchronous section\textsuperscript{436}, and jump down to the failed with attribute step below.}

2. \textit{\textbf{If} urlString and urlRecord be the resulting URL string\textsuperscript{499} and the resulting URL record\textsuperscript{489}, respectively, that would have resulted from parsing\textsuperscript{489} the URL specified by the \texttt{src}\textsuperscript{398} attribute’s value relative to the media element\textsuperscript{388}’s node document when the \texttt{src}\textsuperscript{398} attribute was last changed.}

3. \textit{\textbf{If} urlString was obtained successfully, set the \texttt{currentSrc}\textsuperscript{491} attribute to urlString.}

4. End the synchronous section\textsuperscript{436}, continuing the remaining steps \textbf{in parallel}\textsuperscript{442}.

5. \textit{\textbf{If} urlRecord was obtained successfully, run the resource fetch algorithm\textsuperscript{498} with urlRecord. If that algorithm returns without aborting this one, then the load failed.}

6. \textit{Failed with attribute: Reaching this step indicates that the media resource failed to load or that the given URL could not be parsed\textsuperscript{489}. Take pending play promises\textsuperscript{411} and queue a media element task\textsuperscript{409} given the media element\textsuperscript{388} to run the dedicated media source failure steps\textsuperscript{396} with the result.}

7. Wait for the task\textsuperscript{403} queued by the previous step to have executed.

8. Return. The element won’t attempt to load another resource until this algorithm is triggered again.

\textbf{Otherwise (mode is children)}

1. \textit{\textbf{If} a new node is inserted\textsuperscript{484} 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.

2. \textit{Process candidate: If candidate does not have a \texttt{src}\textsuperscript{319} attribute, or if its \texttt{src}\textsuperscript{310} attribute’s value is the empty string, then end the synchronous section\textsuperscript{436}, and jump down to the failed with elements step below.}

3. \textit{Let urlString and urlRecord be the resulting URL string\textsuperscript{499} and the resulting URL record\textsuperscript{489}, respectively, that would have resulted from parsing\textsuperscript{489} the URL specified by candidate’s \texttt{src}\textsuperscript{319} attribute’s value relative to the candidate’s node document when the \texttt{src}\textsuperscript{319} attribute was last changed.}

4. \textit{If urlString was not obtained successfully, then end the synchronous section\textsuperscript{436}, and jump down to the failed with elements step below.}

5. \textit{If candidate has a \texttt{type}\textsuperscript{318} 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\textsuperscript{291}, then end the synchronous section\textsuperscript{436}, and jump down to the failed with elements step below.}
6. Set the `currentSrc` attribute to `urlString`.

7. End the synchronous section, continuing the remaining steps in parallel.

8. Run the resource fetch algorithm with `urlRecord`. If that algorithm returns without aborting this one, then the load failed.

9. Failed with elements: Queue a media element task given the media element to fire an event named `error` at candidate.

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 media element task given the media element 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 with a list of promises `promises` are the following steps:

1. Set the `error` attribute to the result of creating a `MediaError` with `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 an event named `error` at the media element.

6. Reject pending play promises with promises and a "NotSupportedError" DOMException.

7. 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 URL record or media provider object, is as follows:

1. If the algorithm was invoked with media provider object or a URL record whose blob URL entry is a blob URL entry whose object is a media provider object, then let mode be local. Otherwise let mode be remote.
2. If \textit{mode} is remote, then let the \textit{current media resource} be the resource given by the URL record passed to this algorithm; otherwise, let the \textit{current media resource} be the resource given by the media provider object. Either way, the \textit{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:
   
   \begin{itemize}
   
   \item If \textit{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 \texttt{preload} attribute's \texttt{none} keyword).
   
     1. Set the networkState to \texttt{NETWORK_IDLE}.
   
     2. Queue a media element task given the media element to fire an event named \texttt{suspend} at the element.
   
     3. Queue a media element task given the media element 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 \texttt{NETWORK_LOADING}.
   
   2. Let \textit{destination} be "audio" if the media element is an audio element and to "video" otherwise.

   Let request be the result of creating a potential-CORS request given current media resource's URL record, \textit{destination}, and the media element's \texttt{crossorigin} attribute value.

   Set request's client to the media element's node document's relevant settings object.

   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 \textit{stall timeout} is an implementation-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 media element task given the media element to fire an event named \texttt{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 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 media element task given the media element to set the networkState to \texttt{NETWORK_IDLE} and fire an event named \texttt{suspend} at the element. If and when downloading of the resource resumes, the user agent must queue a media element task given the media element to set the networkState to \texttt{NETWORK_LOADING}.

   Between the queuing of these tasks, the load is suspended (so progress events don't fire, as described above).
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 media element task given the media element 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 (±200ms) or for every byte received, whichever is least frequent, queue a media element task given the media element to fire an 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 media element task given the media element 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 appropriate media element event task source at the element.)

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

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 media element task given the media element 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**[^p389] **is found to have an audio track**

1. Create an `AudioTrack`[^p418] object to represent the audio track.
3. Let `enable` be `unknown`.
4. If either the `media resource[^p389]` or the URL 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**

      This could be triggered by media fragment 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.

5. If `enable` is still `unknown`, then, if the `media element[^p388]` does not yet have an `enabled[^p421]` 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.

→ **If the media resource**[^p389] **is found to have a video track**

1. Create a `VideoTrack[^p418]` object to represent the video track.
3. Let `enable` be `unknown`.
4. If either the `media resource[^p389]` or the URL 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**

      This could again be triggered by media fragment syntax.

5. If `enable` is still `unknown`, then, if the `media element[^p388]` does not yet have a `selected[^p421]` 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.

→ **Once enough of the media data**[^p389] **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[^p403] for the purposes of the current playback position[^p404] and the earliest possible position[^p405], based on the media data[^p389].
2. Update the timeline offset[^p405] to the date and time that corresponds to the zero time in the media.
timeline\(^{389}\) established in the previous step, if any. If no explicit time and date is given by the media resource\(^{389}\), the timeline offset\(^{385}\) must be set to Not-a-Number (NaN).

3. Set the current playback position\(^{404}\) and the official playback position\(^{404}\) to the earliest possible position\(^{405}\).

4. Update the duration\(^{385}\) attribute with the time of the last frame of the resource, if known, on the media timeline\(^{389}\) established above. If it is not known (e.g. a stream that is in principle infinite), update the duration\(^{385}\) attribute to the value positive Infinity.

Note

The user agent will\(^{405}\) queue a media element task\(^{389}\) given the media element\(^{388}\) to fire an event named durationchange\(^{448}\) at the element at this point.

5. For video\(^{380}\) elements, set the videoWidth\(^{383}\) and videoHeight\(^{383}\) attributes, and queue a media element task\(^{389}\) given the media element\(^{388}\) to fire an event named resize\(^{440}\) at the media element\(^{388}\).

Note

Further resize\(^{440}\) events will be fired if the dimensions subsequently change.

6. Set the readyState\(^{408}\) attribute to HAVE_METADATA\(^{346}\).

Note

A loadedmetadata\(^{440}\) DOM event will be fired\(^{407}\) as part of setting the readyState\(^{408}\) attribute to a new value.

7. Let jumped be false.

8. If the media element\(^{388}\)’s default playback start position\(^{404}\) is greater than zero, then seek\(^{416}\) to that time, and let jumped be true.

9. Let the media element\(^{388}\)’s default playback start position\(^{404}\) be zero.

10. Let the initial playback position be zero.

11. If either the media resource\(^{389}\) or the URL 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\(^{416}\) to that time.

Example

For example, with media formats that support media fragment syntax, the fragment can be used to indicate a start position.

12. If there is no enabled\(^{421}\) audio track, then enable an audio track. This will cause a change event to be fired\(^{421}\).

13. If there is no selected\(^{421}\) video track, then select a video track. This will cause a change event to be fired\(^{421}\).

Once the readyState\(^{408}\) attribute reaches HAVE_CURRENT_DATA\(^{406}\), after the loadeddata event has been fired\(^{346}\), set the element’s delaying-the-load-event flag\(^{393}\) to false. This stops delaying the load event\(^{315}\).

Note

A user agent that is attempting to reduce network usage while still fetching the metadata for each media resource\(^{389}\) would also stop buffering at this point, following the rules described previously\(^{397}\), which involve the networkState\(^{392}\) attribute switching to the NETWORK_IDLE\(^{392}\) value and a suspend\(^{439}\) event firing.

Note

The user agent is required to determine the duration of the media resource\(^{389}\) and go through this step before playing.

→ Once the entire media resource\(^{389}\) has been fetched (but potentially before any of it has been decoded) Fire an event named progress\(^{439}\) at the media element\(^{388}\).
Set the `networkState` to `NETWORK_IDLE` and fire an 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 media element task, given the media element 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 the result of creating a `MediaError` with `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 an 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 the result of creating a `MediaError` with `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 an 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 the result of creating a `MediaError` with `MEDIA_ERR_ABORTED`.
3. Fire an 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 an 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.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>State</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>None</td>
<td>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 &quot;play&quot;).</td>
</tr>
<tr>
<td>metadata</td>
<td>Metadata</td>
<td>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 readyState attribute set to HAVE_METADATA, typically though, some frames will be obtained as well and it will probably be HAVE_CURRENT_DATA or HAVE_FUTURE_DATA. 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.</td>
</tr>
<tr>
<td>auto</td>
<td>Automatic</td>
<td>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.</td>
</tr>
</tbody>
</table>

The empty string is also a valid keyword, and maps to the Automatic state. The attribute's missing value default and invalid value default are implementation-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.
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.

User agents may discard previously buffered data.

**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.

**Warning!**

Returning a new object each time is a bad pattern for attribute getters and is only enshrined here as it would be costly to change it. It is not to be copied to new APIs.

### 4.8.12.6 Offsets into the media resource

**For web developers (non-normative)**

#### `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` 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**

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 an implementation-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

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 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.

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`'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 media element task` given the `media element` to `fire an 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 resource` that are before the earliest possible position, the user agent may `queue a media element task` given the `media element` to run these steps:

1. Remove the track from the `audioTracks` attribute's `AudioTrackList` object or the `videoTracks` attribute's `VideoTrackList` object as appropriate.
2. Fire an event named `removetrack` at the `media element`'s aforementioned `AudioTrackList` or `VideoTrackList` object, using `TrackEvent`, with the `track` attribute initialized to the `AudioTrack` or `VideoTrack` object representing the track.

The duration attribute must return the time of the end of the `media resource`, in seconds, on the media timeline. If no `media resource` 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 media element task` given the `media element` to `fire an 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

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.

### 4.8.12.7 Ready states

For web developers (non-normative)

- `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 element's playbackRate 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 will be 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 media element task given the media element to fire an 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 resize 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 media element task given the media element to fire an 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 media element task given the media element to fire an event named `timeupdate` at the element, and queue a media element task given the media element to fire an 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 media element task given the media element to fire an event named `canplay` at the element.

     If the element's `paused` attribute is false, the user agent must notify about playing for 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 media element task given the media element to fire an event named `canplaythrough` at the element.

     If the element is not eligible for autoplay, then the user agent must abort these substeps.

     The user agent may run the following substeps:

     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 media element task given the element to fire an event named `play` at the element.
     4. Notify about playing for the element.

     Alternatively, if the element is a `video` element, the user agent may start observing whether the element intersects the viewport. When the element starts intersecting the viewport, if the element is still eligible for autoplay, run the substeps above. Optionally, when the element stops intersecting the viewport, if the can autoplay flag is still true and the `autoplay` attribute is still specified, run the following substeps:

     1. Run the `internal pause steps` and set the `can autoplay flag` to true.
     2. Queue a media element task given the element to fire an event named `pause` at the element.
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.

The `autoplay` IDL attribute must reflect the content attribute of the same name.

4.8.12.8 Playing the media resource

For web developers (non-normative)

- `media.paused` [ = value ]
  Returns true if playback is paused; false otherwise.

- `media.ended` [ = value ]
  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.preservesPitch`
  Returns true if pitch-preserving algorithms are used when the playbackRate is not 1.0. The default value is true.
  Can be set to false to have the media resource's audio pitch change up or down depending on the playbackRate. This is useful for aesthetic and performance reasons.

- `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...
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 be **eligible for autoplay** when all of the following conditions are met:

- Its **can autoplay** flag is true.
- Its **paused** attribute is true.
- It has an **autoplay** attribute specified.
- Its **node document's active sandboxing flag** does not have the **sandboxed automatic features browsing context flag** set.
- Its **node document** is **allowed to use** the **autoplay** feature.

A **media element** is said to be **allowed to play** if the user agent and the system allow media playback in the current context.

**Example**

For example, a user agent could allow playback only when the **media element's Window object** has **transient activation**, but an exception could be made to allow playback while **muted**.

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.

```javascript
media.pause()
Sets the **paused** attribute to true, loading the **media resource** if necessary.
```
When a media element stops playing because it has paused for user interaction, the user agent must queue a media element task given the media element to fire an 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 nonzero 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

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 return.
2. As defined above, the ended IDL attribute starts returning true once the event loop returns to step 1.
3. Queue a media element task given the media element and the following steps:
   1. Fire an event named timeupdate at the media element.
   2. If the media element has ended playback, the direction of playback is forwards, and paused is false, then:
      1. Set the paused attribute to true.
      2. Fire an event named pause at the media element.
      3. Take pending play promises and reject pending play promises with the result and an "AbortError" DOMException.
   3. Fire an 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 media element task given the media element to fire an 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 user agent must follow these steps:

1. If the given value is not supported by the user agent, then throw a "NotSupportedException" DOMException.
2. Set playbackRate to the new value, and if the element is potentially playing, change the playback speed.
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 media element task given the media element to fire an event named `ratechange` at the media element. The user agent must process attribute changes smoothly and must not introduce any perceivable gaps or muting of playback in response.

The `preservesPitch` getter steps are to return true if a pitch-preserving algorithm is in effect during playback. The setter steps are to correspondingly switch the pitch-preserving algorithm on or off, without any perceivable gaps or muting of playback. By default, such a pitch-preserving algorithm must be in effect (i.e., the getter will initially return true).

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.

⚠ Warning! Returning a new object each time is a bad pattern for attribute getters and is only enshrined here as it would be costly to change it. It is not to be copied to new APIs.

Each media element has a list of pending play promises, which must initially be empty.

To take pending play promises for a media element, the user agent must run the following steps:

1. Let promises be an empty list of promises.
2. Copy the media element's list of pending play promises to promises.
3. Clear the media element's list of pending play promises.
4. Return promises.

To resolve pending play promises for a media element with a list of promises promises, the user agent must resolve each promise in promises with undefined.

To reject pending play promises for a media element with a list of promise promises and an exception name error, the user agent must reject each promise in promises with error.

To notify about playing for a media element, the user agent must run the following steps:

1. Take pending play promises and let promises be the result.
2. Queue a media element task given the element and the following steps:
   1. Fire an event named `playing` at the element.
   2. Resolve pending play promises with promises.

When the `play()` method on a media element is invoked, the user agent must run the following steps:

1. If the media element is not allowed to play, return a promise rejected with a "NotAllowedError" DOMException.
2. If the media element's error attribute is not null and its code is `MEDIA_ERR_SRC_NOT_SUPPORTED`, return a promise rejected with a "NotFoundError" DOMException.

Note This means that the dedicated media source failure steps have run. Playback is not possible until the media element load algorithm clears the error attribute.

3. Let promise be a new promise and append promise to the list of pending play promises.
4. Run the internal play steps for the media element.
5. Return promise.

The internal play steps for a media element are as follows:
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 media element task given the media element to fire an event named `timeupdate` at the media element.

3. If the `media element`'s `paused` attribute is true, then:
   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 media element task given the `media element` to fire an 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 media element task given the `media element` to fire an event named `waiting` at the element.

   Otherwise, the `media element`'s `readyState` attribute has the value `HAVE FUTURE_DATA` or `HAVE ENOUGH_DATA`: notify about playing for the element.

4. Otherwise, if the `media element`'s `readyState` attribute has the value `HAVE FUTURE_DATA` or `HAVE ENOUGH_DATA`, take pending play promises and queue a media element task given the `media element` to resolve pending play promises with the result.

5. Set the `media element`'s `can autoplay 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 `can autoplay 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. Take pending play promises and let `promises` be the result.
   3. Queue a media element task given the `media element` and the following steps:
      1. Fire an event named `timeupdate` at the element.
      2. Fire an event named `pause` at the element.
      3. Reject pending play promises with `promises` and an "AbortError" DOMException.
   4. Set the official playback position to the current playback position.

If the element's `playbackRate` 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, its current playback position must increase monotonically at the element’s playbackRate 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 element’s playbackRate is negative.)

**Note**
The element’s playbackRate 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).

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 element’s playbackRate is so low or so high that the user agent cannot play audio usefully, the corresponding audio must also be muted. If the element’s playbackRate is not 1.0 and preservesPitch is true, the user agent must apply pitch adjustment to preserve the original pitch of the audio. Otherwise, the user agent must speed up or slow down the audio without any pitch adjustment.

When a media element is potentially playing, its audio data played must be 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 a media element is not potentially playing, audio must not play for the element.

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, it could be unpaused but stilled waiting for content to buffer, or it could be still buffering, but with a suspend event listener that begins playback. Even 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 for 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. To support use cases that depend on the timing accuracy of cue event firing, such as synchronizing captions with shot changes in a video, user agents should fire cue events as close as possible to their position on the media timeline, and ideally within 20 milliseconds. 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.

**Note**
If one iteration takes a long time, this can cause short duration cues to be skipped over as the user agent rushes ahead to “catch up”, so these cues will not appear in the activeCues list.
The **time marches on** steps are as follows:

1. Let **current cues** be a list of cues\(^{424}\), initialized to contain all the cues\(^{424}\) of all the hidden\(^{423}\) or showing\(^{423}\) text tracks\(^{422}\) of the media element\(^{388}\) (not the disabled\(^{422}\) ones) whose start times\(^{424}\) are less than or equal to the current playback position\(^{404}\) and whose end times\(^{424}\) are greater than the current playback position\(^{404}\).

2. Let **other cues** be a list of cues\(^{424}\), initialized to contain all the cues\(^{424}\) of hidden\(^{423}\) and showing\(^{423}\) text tracks\(^{422}\) of the media element\(^{388}\) that are not present in **current cues**.

3. Let **last time** be the current playback position\(^{404}\) at the time this algorithm was last run for this media element\(^{388}\), if this is not the first time it has run.

4. If the current playback position\(^{404}\) 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\(^{424}\) in other cues whose start times\(^{424}\) are greater than or equal to last time and whose end times\(^{424}\) are less than or equal to the current playback position\(^{404}\). Otherwise, let missed cues be an empty list.

5. Remove all the cues\(^{424}\) in missed cues that are also in the media element\(^{388}\)’s list of newly introduced cues\(^{413}\), and then empty the element’s list of newly introduced cues\(^{413}\).

6. If the time was reached through the usual monotonic increase of the current playback position\(^{404}\) during normal playback, and if the user agent has not fired a timeupdate\(^{416}\) 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 media element task\(^{438}\) given the media element\(^{388}\) to fire an event named timeupdate\(^{416}\) 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\(^{404}\).)

**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\(^{424}\) in current cues have their text track cue active flag\(^{424}\) set, none of the cues\(^{424}\) in other cues have their text track cue active flag\(^{424}\) set, and missed cues is empty, then return.

8. If the time was reached through the usual monotonic increase of the current playback position\(^{404}\) during normal playback, and there are cues\(^{424}\) in other cues that have their text track cue pause-on-exit flag\(^{424}\) set and that either have their text track cue active flag\(^{424}\) set or are also in missed cues, then immediately\(^{42}\) pause\(^{42}\) the media element\(^{388}\).

**Note**
In the other cases, such as explicit seeks, playback is not paused by going past the end time of a cue\(^{424}\), even if that cue\(^{424}\) has its text track cue pause-on-exit flag\(^{424}\) set.

9. Let events be a list of tasks\(^{431}\), initially empty. Each task\(^{431}\) in this list will be associated with a text track\(^{422}\), a text track cue\(^{42}\), and a time, which are used to sort the list before the tasks\(^{431}\) are queued.

Let affected tracks be a list of text tracks\(^{422}\), initially empty.

When the steps below say to prepare an event named event for a text track cue\(^{42}\) target with a time time, the user agent must run these steps:

1. Let track be the text track\(^{42}\) with which the text track cue\(^{42}\) target is associated.

2. Create a task\(^{431}\) to fire an event named event at target.

3. Add the newly created task\(^{431}\) to events, associated with the time time, the text track\(^{42}\) track, and the text track cue\(^{42}\) target.

4. Add track to affected tracks.

10. For each text track cue\(^{42}\) in missed cues, prepare an event\(^{414}\) named enter\(^{441}\) for the TextTrackCue\(^{434}\) object with the text track cue start time\(^{42}\).

11. For each text track cue\(^{42}\) in other cues that either has its text track cue active flag\(^{42}\) set or is in missed cues, prepare an event\(^{414}\) named exit\(^{441}\) for the TextTrackCue\(^{434}\) object with the later of the text track cue end time\(^{42}\) and the text track cue start time\(^{42}\).
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 a media element task given the media element for 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 media element task given the media element to fire an event named cuechange at the TextTrack object, and, if the text track has a corresponding track element, to then fire an event named cuechange at the track element as well.

17. Set the text track active flag of all the cues in the current cues, and unset the text track 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.

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, return.

3. ⌛ Run the internal pause steps for the media element.

**4.8.12.9 Seeking**

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 is readyState is HAVE NOTHING, return.
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 return.
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

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 media element task given the media element to fire an 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 media element task given the media element to fire an event named timeupdate at the element.

17. Queue a media element task given the media element to fire an event named seeked at the element.

The seekable attribute must return a new static normalized TimeRanges object that represents the ranges of the media resource, 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).

- 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.

**⚠ Warning!**

Returning a new object each time is a bad pattern for attribute getters and is only enshrined here as it would be costly to change it. It is not to be copied to new APIs.

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**

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).

4.8.12.10 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 dialogues, director’s commentaries, audio descriptions, alternative angles, or sign-language overlays.

**For web developers (non-normative)**

- `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.
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.)

4.8.12.10.1 AudioTrackList and VideoTrackList objects

The AudioTrackList and VideoTrackList interfaces are used by attributes defined in the previous section.

```idl
[Exposed=Window]
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;
};

[Exposed=Window]
interface AudioTrack {
  readonly attribute DOMString id;
  readonly attribute DOMString kind;
  readonly attribute DOMString label;
  readonly attribute DOMString language;
  attribute boolean enabled;
};

[Exposed=Window]
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;
};

[Exposed=Window]
interface VideoTrack {
  readonly attribute DOMString id;
  readonly attribute DOMString kind;
  readonly attribute DOMString label;
  readonly attribute DOMString language;
  attribute boolean selected;
};
```

For web developers (non-normative)

```javascript
media.audioTracks.length
media.videoTracks.length
```

Returns the number of tracks in the list.

```javascript
audioTrack = media.audioTracks[index]
videoTrack = media.videoTracks[index]
```

Returns the specified AudioTrack or VideoTrack object.
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.

The `AudioTrackList.length` and `VideoTrackList.length` attribute getters 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...
The primary audio or video track.

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 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 media fragment 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.

Example

For example, in Ogg files, this would be the Name header field of the track. [OGGSKELETONHEADERS]

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 only must be returned for VideoTrack objects if they are appropriate for video.

For Ogg files, the Role header field of the track gives the relevant metadata. For DASH media resources, the Role element conveys the information. For WebM, only the FlagDefault element currently maps to a value. Sourcing In-band Media Resource Tracks from Media Containers into HTML has further details. [OGGSKELETONHEADERS] [DASH] [WEBMCG] [INBAND]

<table>
<thead>
<tr>
<th>Category</th>
<th>Definition</th>
<th>Applies to...</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;alternative&quot;</td>
<td>A possible alternative to the main track, e.g. a different take of a song</td>
<td>Audio and video.</td>
<td>Ogg: &quot;audio/alternate&quot; or &quot;video/alternate&quot;; DASH: &quot;alternate&quot; without &quot;main&quot; and &quot;commentary&quot; roles, and, for audio, without the &quot;dub&quot; role (other roles ignored).</td>
</tr>
<tr>
<td></td>
<td>(audio), or a different angle (video).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;captions&quot;</td>
<td>A version of the main video track with captions burnt in. (For legacy</td>
<td>Video only.</td>
<td>DASH: &quot;caption&quot; and &quot;main&quot; roles together (other roles ignored).</td>
</tr>
<tr>
<td></td>
<td>content; new content would use text tracks.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;descriptions&quot;</td>
<td>An audio description of a video track.</td>
<td>Audio only.</td>
<td>Ogg: &quot;audio/audiodesc&quot;.</td>
</tr>
<tr>
<td>&quot;main&quot;</td>
<td>The primary audio or video track.</td>
<td>Audio and video.</td>
<td>Ogg: &quot;audio/main&quot; or &quot;video/main&quot;; WebM: the FlagDefault element is set; DASH: &quot;main&quot; role without &quot;caption&quot;, &quot;subtitle&quot;, and &quot;dub&quot; roles (other roles ignored).</td>
</tr>
<tr>
<td>&quot;main-desc&quot;</td>
<td>The primary audio track, mixed with audio descriptions.</td>
<td>Audio only.</td>
<td>AC3 audio in MPEG-2 TS: bsmod=2 and full_svc=1.</td>
</tr>
<tr>
<td>&quot;sign&quot;</td>
<td>A sign-language interpretation of an audio track.</td>
<td>Video only.</td>
<td>Ogg: &quot;video/sign&quot;.</td>
</tr>
<tr>
<td>&quot;subtitles&quot;</td>
<td>A version of the main video track with subtitles burnt in. (For legacy</td>
<td>Video only.</td>
<td>DASH: &quot;subtitle&quot; and &quot;main&quot; roles together (other roles ignored).</td>
</tr>
<tr>
<td></td>
<td>content; new content would use text tracks.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;translation&quot;</td>
<td>A translated version of the main audio track.</td>
<td>Audio only.</td>
<td>Ogg: &quot;audio/dub&quot;. DASH: &quot;dub&quot; and &quot;main&quot; roles together (other roles ignored).</td>
</tr>
<tr>
<td>&quot;commentary&quot;</td>
<td>Commentary on the primary audio or video track, e.g. a director's</td>
<td>Audio and video.</td>
<td>DASH: &quot;commentary&quot; role without &quot;main&quot; role (other roles ignored).</td>
</tr>
<tr>
<td></td>
<td>commentary.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;&quot; (empty string)</td>
<td>No explicit kind, or the kind given by the track's metadata is not</td>
<td>Audio and video.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>recognized by the user agent.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The `AudioTrack` label and `VideoTrack` label attributes must return the label of the track, if it has one, or the empty string otherwise. [INBAND]

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. [INBAND]

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 media element task, given the media element to fire an 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 media element task given the media element to fire an event named change at the `VideoTrackList` object. This task must be queued before 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 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

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:

<table>
<thead>
<tr>
<th>Event handler</th>
<th>Event handler event type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>onchange</code></td>
<td><code>change</code></td>
</tr>
<tr>
<td><code>onaddtrack</code></td>
<td><code>addtrack</code></td>
</tr>
<tr>
<td><code>onremovetrack</code></td>
<td><code>removetrack</code></td>
</tr>
</tbody>
</table>

4.8.12.10.2 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 of the URL of the media resource. The format of the fragment depends on the MIME type of the media resource, [RFC2046] [URL].
In this example, a video that uses a format that supports media fragment syntax is embedded in such a way that the alternative angles labeled "Alternative" are enabled instead of the default video track.

```html
<video src="myvideo#track=Alternative"></video>
```

4.8.12.11 Timed text tracks

4.8.12.11.1 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's format specification), 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**

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 parsed, 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’s 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 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, return.
2. Set the media element’s pending text track change notification flag.
3. Queue a media element task given the media element to run these steps:
   1. Unset the media element’s pending text track change notification flag.
   2. Fire an 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, including the actual data of the cue. For example, WebVTT has a text track cue writing direction and so forth. [WEBVTT]

---

**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 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. 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 synchronously unset this flag whenever the text track cue's is removed from its text track's 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].

4.8.12.11.2 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 Sourcing In-band Media Resource Tracks from Media Containers into HTML, [INBAND].

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 by the relevant specification. 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 chapters or 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 Name header field. [OGGSKELHEADERS]

   - 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. [WEBMCG]

   - 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 Generic coding of moving pictures and associated audio information. 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, expressed in hexadecimal using ASCII upper hex digits. [MPEG2]

   - 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

425
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]

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.

8. Add the new text track to the media element's list of text tracks.

9. Fire an event named addtrack at the media element's TextTrackList object, using TrackEvent, with the track attribute initialized to the text track's TextTrack object.

4.8.12.11.3 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:

<table>
<thead>
<tr>
<th>State</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subtitles</td>
<td>subtitles</td>
</tr>
<tr>
<td>Captions</td>
<td>captions</td>
</tr>
<tr>
<td>Descriptions</td>
<td>descriptions</td>
</tr>
<tr>
<td>Chapters metadata</td>
<td>chapters</td>
</tr>
<tr>
<td>Metadata</td>
<td>metadata</td>
</tr>
</tbody>
</table>

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 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 media element task given the media element's TextTrackList object, using TrackEvent, with the track attribute initialized to the text track's TextTrack 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 media element task given the media element to fire an event named `removetrack` at the `media element`'s `TextTrackList` object, using `TrackEvent`, with the `track` attribute initialized to the `text track`'s `TextTrack` 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 media element task given the media element to run the following steps for the `media element`:

1. If the element's `blocked-on-parser` flag is true, then return.
2. If the element's `did-perform-automatic-track-selection` flag is true, then return.
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. If there are any `text tracks` in the `media element`'s list of text tracks whose `text track kind` is `chapters` or `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`.
4. 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 return.
3. If any of the `text tracks` in `candidates` have a `text track mode` set to `showing`, return.
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 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, return, 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`, then return.
3. If the text track's track element does not have a media element as a parent, return.

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 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, then:

    1. Let request be the result of creating a potential-CORS request given URL, "track", and corsAttributeState, and with the same-origin fallback flag set.

    2. Set request's client to the track element's node document's relevant settings object.

    3. Fetch request.

The tasks 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 specifications' 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 fetching fails for any reason (network error, the server returns an error code, CORS fails, etc), or if URL is the empty string, then queue an element task on the DOM manipulation task source given the media element to first change the text track readiness to failed to load and then fire an event named error at the track element.

If fetching 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 XML requires be detected and reported to the application), then the task that is queued on the networking task source in which the aforementioned problem is found must change the text track readiness to failed to load and fire an event named error at the track element.

If fetching does not fail, and the file was successfully processed, then the final task by the networking task source, after it has finished parsing the data, must change the text track readiness state to loaded and fire an event named load at the track element.

If, while fetching is ongoing, either:

- the track URI 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 an element task on the DOM manipulation task source given the track element that first changes the text track readiness state to failed to load and then fires an event named error at the track element.
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 showing.

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.)

### 4.8.12.11.4 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. 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 analogue to a per-cue identifier.

**The text track cue pause-on-exit flag**

Should be set to false.

### 4.8.12.11.5 Text track API

```idl
[Exposed=Window]
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;
};
```

For web developers (non-normative)

**media . textTracks**

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 nth text track in the media element’s list of text tracks.

**textTrack = media . textTracks[n] . 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, the user agent must return the indexth 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 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.

```idl
enum TextTrackMode { "disabled", "hidden", "showing" };
enum TextTrackKind { "subtitles", "captions", "descriptions", "chapters", "metadata" };

[Exposed=Window]
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;

    undefined addCue(TextTrackCue cue);
    undefined removeCue(TextTrackCue cue);

    attribute EventHandler oncuechange;
};
```

For web developers (non-normative)

```javascript
let 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 if the format supports media fragment syntax, and that can be used with the `getTrackById(id)` method.

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.
```
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 media element task given the media element to fire an event named `addtrack` at the media element's `TextTrackList` object, using `TrackEvent`, with the `track` attribute initialized to the new text track's `TextTrack` 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 media fragment 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.

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 the 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" DOMException.
3. If the given cue is in the text track list of cues, then remove cue from that text track list of cues.
4. Add cue to the TextTrack object'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 in the TextTrack object's text track list of cues, then throw a "NotFoundError" DOMException.
2. Remove cue from the TextTrack object's text track list of cues.

Example

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.

```javascript
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,
// and have the browser ask them to stay
window.onbeforeunload = function (e) {
    playSound('kitten mew');
e.preventDefault();
}

IDL [Exposed=Window]
interface TextTrackCueList {
    readonly attribute unsigned long length;
    getter TextTrackCue (unsigned long index);
    TextTrackCue? getCueById(DOMString id);
};

For web developers (non-normative)

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 indexth 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.
IDL

```idl
[Exposed=Window]
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;
};
```

For web developers (non-normative)

- **cue . track**
  Returns the `TextTrack` object to which this text track cue belongs, if any, or null otherwise.

- **cue . id**
  Returns the text track cue identifier. Can be set.

- **cue . startTime**
  Returns the text track cue start time, in seconds. Can be set.

- **cue . endTime**
  Returns the text track cue end time, in seconds. Can be set.

- **cue . pauseOnExit**
  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` whose list of cues the `text track cue` 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` is set, 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.

### 4.8.12.11.6 Event handlers for objects of the text track APIs

The following are the `event handlers` that (and their corresponding `event handler event types`) that must be supported, as `event handler IDL attributes`, by all objects implementing the `TextTrackList` interface:

- **cue**
  - `track`
  - `id`
  - `startTime`
  - `endTime`
  - `pauseOnExit`
The following are the event handlers that (and their corresponding event handler event types) that must be supported, as event handler IDL attributes, by all objects implementing the TextTrack interface:

<table>
<thead>
<tr>
<th>Event handler</th>
<th>Event handler event type</th>
</tr>
</thead>
<tbody>
<tr>
<td>onchange</td>
<td>change</td>
</tr>
<tr>
<td>onaddtrack</td>
<td>addtrack</td>
</tr>
<tr>
<td>onremovetrack</td>
<td>removetrack</td>
</tr>
</tbody>
</table>

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 TextTrackCue interface:

<table>
<thead>
<tr>
<th>Event handler</th>
<th>Event handler event type</th>
</tr>
</thead>
<tbody>
<tr>
<td>oncuechange</td>
<td>cuechange</td>
</tr>
<tr>
<td>onenter</td>
<td>enter</td>
</tr>
<tr>
<td>onexit</td>
<td>exit</td>
</tr>
</tbody>
</table>

4.8.12.11.7 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 overlayed with the scores, as follows:

![Image of a robotics competition with scores and a timestamp]

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.)

4.8.12.12 Identifying a track kind through a URL

Other specifications or formats that need a URL to identify the return values of the `AudioTrack` or `VideoTrack` kind IDL attributes, or identify the kind of text track, must use the `about:html-kind` URL.

4.8.12.13 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.

Even when the attribute is absent, however, user agents 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.

Features such as fast-forward or rewind must be implemented by only changing the playbackRate attribute (and not the defaultPlaybackRate attribute).

Seeking must be implemented in terms of seeking to the requested position in the media element's media timeline. For media resources where seeking to an arbitrary position would be slow, user agents are encouraged to use the approximate-for-speed flag when seeking in response to the user manipulating an approximate position interface such as a seek bar.

The controls IDL attribute must reflect the content attribute of the same name.

For web developers (non-normative)

```javascript
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" DOMException 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" DOMException 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 media element task given the media element to fire an event named volumechange at the media element. Then, if the media element is not allowed to play, the user agent must run the internal pause steps for 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 return the volume desired by the user.
2. If the element’s audio output is muted, then return zero.
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. Return 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**

This video (an advertisement) autoplays, but to avoid annoying users, it does so without sound, and allows the user to turn the sound on. The user agent can pause the video if it's unmuted without a user interaction.

```html
<video src="adverts.cgi?kind=video" controls autoplay loop muted></video>
```

### 4.8.12.14 Time ranges

Objects implementing the TimeRanges interface represent a list of ranges (periods) of time.

```idl
[Exposed=Window]
interface TimeRanges {
    readonly attribute unsigned long length;
    double start(unsigned long index);
    double end(unsigned long index);
};
```

**For web developers (non-normative)**

- `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" DOMException 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" DOMException 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 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 range represented by the object, in seconds measured from the start of the timeline that the object covers.

These methods must throw "IndexSizeError" DOMExceptions 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

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.

4.8.12.15 The TrackEvent interface

```idl
[Exposed=Window]
interface TrackEvent : Event {
    constructor(DOMString type, optional TrackEventInit eventInitDict = {});
    readonly attribute (VideoTrack or AudioTrack or TextTrack)? track;
};
dictionary TrackEventInit : EventInit {
    (VideoTrack or AudioTrack or TextTrack)? track = null;
};
```

For web developers (non-normative)

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. It represents the context information for the event.

4.8.12.16 Events summary

This section is non-normative.

The following events fire on media elements, as part of the processing model described above:

<table>
<thead>
<tr>
<th>Event name</th>
<th>Interface</th>
<th>Fired when...</th>
<th>Preconditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>loadstart</td>
<td>Event</td>
<td>The user agent begins looking for media data, as part of the resource selection algorithm.</td>
<td>networkState equals NETWORK_LOADING.</td>
</tr>
<tr>
<td>progress</td>
<td>Event</td>
<td>The user agent is fetching media data.</td>
<td>networkState equals NETWORK_LOADING.</td>
</tr>
<tr>
<td>suspend</td>
<td>Event</td>
<td>The user agent is intentionally not currently fetching media data.</td>
<td>networkState equals NETWORK_IDLE.</td>
</tr>
<tr>
<td>abort</td>
<td>Event</td>
<td>The user agent stops fetching the media data before it is completely downloaded, but not due to an error.</td>
<td>error is an object with the code MEDIA_ERR_ABORTED, networkState equals either NETWORK_EMPTY or NETWORK_IDLE, depending on when the download was aborted.</td>
</tr>
<tr>
<td>error</td>
<td>Event</td>
<td>An error occurs while fetching the media data or the type of the resource is not supported media format.</td>
<td>error is an object with the code MEDIA_ERR_NETWORK or higher. networkState equals either NETWORK_EMPTY or NETWORK_IDLE, depending on when the download was aborted.</td>
</tr>
<tr>
<td>emptied</td>
<td>Event</td>
<td>A media element whose networkState was previously not in the NETWORK_EMPTY state has just switched to that state (either because of a fatal error during load that's about to be reported, or because the networkState is NETWORK_EMPTY; all the IDL attributes are in their initial states.)</td>
<td></td>
</tr>
<tr>
<td>Event name</td>
<td>Interface</td>
<td>Fired when...</td>
<td>Preconditions</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>load()</td>
<td>Event</td>
<td><code>load()</code> method was invoked while the resource selection algorithm was already running.</td>
<td></td>
</tr>
<tr>
<td>stalled</td>
<td>Event</td>
<td>The user agent is trying to fetch media data, but data is unexpectedly not forthcoming.</td>
<td>networkState is <code>NETWORK_LOADING</code>.</td>
</tr>
<tr>
<td>loadedmetadata</td>
<td>Event</td>
<td>The user agent has determined the duration and dimensions of the media resource and the text tracks are ready.</td>
<td>readyState is newly equal to <code>HAVE_METADATA</code> or greater for the first time.</td>
</tr>
<tr>
<td>loadeddata</td>
<td>Event</td>
<td>The user agent can render the media data at the current playback position for the first time.</td>
<td>readyState newly increased to <code>HAVE_CURRENT_DATA</code> or greater for the first time.</td>
</tr>
<tr>
<td>canplay</td>
<td>Event</td>
<td>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.</td>
<td>readyState newly increased to <code>HAVE_FUTURE_DATA</code> or greater.</td>
</tr>
<tr>
<td>canplaythrough</td>
<td>Event</td>
<td>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.</td>
<td>readyState newly equal to <code>HAVE_ENOUGH_DATA</code>.</td>
</tr>
</tbody>
</table>
| 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_CURRENT_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.

<table>
<thead>
<tr>
<th>Event name</th>
<th>Interface</th>
<th>Fired when...</th>
<th>Preconditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>waiting</td>
<td>Event</td>
<td>Playback has stopped because the next frame is not available, but the user agent expects that frame to become available in due course.</td>
<td>readyState is equal to or less than <code>HAVE_CURRENT_DATA</code>, and <code>paused</code> is false. Either <code>seeking</code> is true, or the <code>current playback position</code> is not contained in any of the ranges in <code>buffered</code>. It is possible for playback to stop for other reasons without <code>paused</code> being false, but those reasons do not fire this event (and when those situations resolve, a separate <code>playing</code> event is not fired either). e.g. <code>playback has ended</code> or <code>playback stopped due to errors</code>, or the element has paused for user interaction or paused for in-band content.</td>
</tr>
<tr>
<td>seeking</td>
<td>Event</td>
<td>The <code>seeking</code> IDL attribute changed to true, and the user agent has started seeking to a new position.</td>
<td></td>
</tr>
<tr>
<td>sought</td>
<td>Event</td>
<td>The <code>seeking</code> IDL attribute changed to false after the <code>current playback position</code> was changed.</td>
<td></td>
</tr>
<tr>
<td>ended</td>
<td>Event</td>
<td>Playback has stopped because the end of the media resource was reached.</td>
<td><code>currentTime</code> equals the end of the media resource; <code>ended</code> is true.</td>
</tr>
<tr>
<td>durationchange</td>
<td>Event</td>
<td>The <code>duration</code> attribute has just been updated.</td>
<td></td>
</tr>
<tr>
<td>timeupdate</td>
<td>Event</td>
<td>The <code>current playback position</code> changed as part of normal playback or in an especially interesting way, for example discontinuously.</td>
<td></td>
</tr>
<tr>
<td>play</td>
<td>Event</td>
<td>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.</td>
<td><code>paused</code> is newly false.</td>
</tr>
<tr>
<td>pause</td>
<td>Event</td>
<td>The element has been paused. Fired after the <code>pause()</code> method has returned.</td>
<td><code>paused</code> is newly true.</td>
</tr>
<tr>
<td>ratechange</td>
<td>Event</td>
<td>Either the <code>defaultPlaybackRate</code> or the <code>playbackRate</code> attribute has just been updated.</td>
<td></td>
</tr>
<tr>
<td>resize</td>
<td>Event</td>
<td>One or both of the <code>videoWidth</code> and <code>videoHeight</code> attributes have just been updated.</td>
<td><code>Media element</code> is a <code>video</code> element; <code>readyState</code> is not <code>HAVE NOTHING</code>.</td>
</tr>
<tr>
<td>volumechange</td>
<td>Event</td>
<td>Either the <code>volume</code> attribute or the <code>muted</code> attribute has changed. Fired after the relevant attribute's setter has returned.</td>
<td></td>
</tr>
</tbody>
</table>

The following event fires on `source` element:
### 4.8.12.17 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.

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 more or less be obtained using the `img` 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. That information is therefore only exposed if the video resource uses CORS. The `crossorigin` attribute allows authors to enable CORS.

Example

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.

Example

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.
4.8.12.18 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.

4.8.12.19 Best practices for implementers 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-bitrate 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.)

4.8.13 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.
Accessibility considerations:

For authors.
For implementers.

DOM interface:

```
[Exposed=Window]
interface HTMLMapElement : HTMLElement {
    [HTMLConstructor] constructor();

    [CEReactions] attribute DOMString name;
    [SameObject] readonly attribute HTMLCollection areas;
};
```

The `map` element, in conjunction with an `img` element and any `area` element descendants, defines an `imagemap`. 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 ASCII whitespace. The value of the `name` attribute must not be equal to the value of the `name` attribute of another `map` element in the same tree. If the `id` attribute is also specified, both attributes must have the same value.

For web developers (non-normative)

```
map . areas

Returns an HTMLCollection of the area elements in the map.
```

The `areas` attribute must return an HTMLCollection rooted at the `map` element, whose filter matches only area elements.

The IDL attribute `name` must reflect the content attribute of the same name.

Example

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>
<html LANG="EN">
  <head>
    <title>Babies™: Toys</title>
    <body>
      <h1>Toys</h1>
      <img SRC="/images/menu.gif"
           ALT="Babies™ navigation menu. Select a department to go to its page."
           USEMAP="#NAV">
    </body>
    <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>
  </footer>
```

443
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.

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, when presented with the texts specified for the other hyperlinks of the **image map**, and with the alternative text of the image, but without the image itself, provides the user with the same kind of choice as the hyperlink would when used without its text but with its shape applied to the image. The **alt** attribute may be left blank if there is another **area** element in the same **image map** that points to the same resource and has a non-blank **alt** attribute.

If the **area** element has no **href** attribute, then the area represented by the element cannot be selected, and the **alt** attribute...
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.

<table>
<thead>
<tr>
<th>State</th>
<th>Keywords</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circle state</td>
<td>circle</td>
<td></td>
</tr>
<tr>
<td></td>
<td>circ</td>
<td>Non-conforming</td>
</tr>
<tr>
<td>Default state</td>
<td>default</td>
<td></td>
</tr>
<tr>
<td>Polygon state</td>
<td>poly</td>
<td></td>
</tr>
<tr>
<td></td>
<td>polygon</td>
<td>Non-conforming</td>
</tr>
<tr>
<td>Rectangle state</td>
<td>rect</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rectangle</td>
<td>Non-conforming</td>
</tr>
</tbody>
</table>

The attribute may be omitted. The `missing value default` and `invalid value default` are 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 from the top edge 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, `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`, `download`, and `ping` attributes decide how the link is followed. The `rel` attribute may be used to indicate to the user the likely nature of the target resource before the user follows the link.

The `target`, `download`, `ping`, `rel`, and `referrerpolicy` attributes must be omitted if the `href` attribute is not present.

If the `itemprop` attribute is specified on an `area` element, then the `href` attribute must also be specified.

The activation behavior of `area` elements is to 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`, `ping`, and `rel`, 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 IDL attribute `referrerPolicy` must `reflect` the `referrerPolicy` content attribute, limited to only known values.
4.8.15 Image maps

4.8.15.1 Authoring

An image map allows geometric areas on an image to be associated with hyperlinks.

An image, in the form of an img element or an object element representing an image, may be associated with an image map (in the form of a map element) by specifying a usemap attribute on the img or object element. The usemap attribute, if specified, must be a valid hash-name reference to a map element.

Example

Consider an image that looks as follows:

If we wanted just the colored areas to be clickable, we could do it as follows:

```html
<p>
Please select a shape:
<img src="shapes.png" usemap="#shapes"
    alt="Four shapes are available: a red hollow box, a green circle, a blue triangle, and a yellow four-pointed star."/>
<map name="shapes">
  <area shape=rect coords="50,50,100,100" href="red.html" alt="Red box."/>
  <area shape=rect coords="25,25,125,125" href="green.html" alt="Green circle.">
  <area shape=circle coords="200,75,50" href="blue.html" alt="Blue triangle.">
  <area shape=poly coords="325,25,262,125,388,125" href="yellow.html" alt="Yellow star.">
</map>
</p>
```

4.8.15.2 Processing model

If an img element or an object element representing an image 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 as the context node. This will return either an element (the map) or null.
2. If that returned null, then return. 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 an implementation-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; return.

<table>
<thead>
<tr>
<th>State</th>
<th>Minimum number of items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circle state</td>
<td>3</td>
</tr>
<tr>
<td>Default state</td>
<td>0</td>
</tr>
<tr>
<td>Polygon state</td>
<td>6</td>
</tr>
<tr>
<td>Rectangle state</td>
<td>4</td>
</tr>
</tbody>
</table>

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; return.

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 $r$ CSS pixels.

   - **Default state**: The shape is a rectangle that exactly covers the entire image.
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][p2273]

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).

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.

4.8.16 MathML

The MathML math element falls into the embedded content, phrasing content, flow content, and palpable content categories for the purposes of the content models in this specification.

When the MathML annotation-xml element contains elements from the HTML namespace, such elements must all be flow content.

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.

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 a MathML mtext element. (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 a MathML merror element containing some appropriate error message.

The semantics of MathML elements are defined by MathML and other applicable specifications. [MATHML]

Example

Here is an example of the use of MathML in an HTML document:

```html
<!DOCTYPE html>
<html lang="en">
  
</html>
```
4.8.17 SVG

The SVG `svg` element falls into the embedded content\(^ {133}\), phrasing content\(^ {133}\), flow content\(^ {132}\), and palpable content\(^ {134}\) categories for the purposes of the content models in this specification.

When the SVG `foreignObject` element contains elements from the HTML namespace, such elements must all be flow content\(^ {132}\).

The content model for the SVG `title` element inside HTML documents is phrasing content\(^ {133}\). (This further constrains the requirements given in SVG 2.)

The semantics of SVG elements are defined by SVG 2 and other applicable specifications\(^ {665}\). [SVG]\(^ {1276}\)

For web developers (non-normative)

```javascript
doc = iframe . getSVGDocument()  
doc = embed . getSVGDocument()  
doc = object . getSVGDocument()  
```

Returns the Document\(^ {115}\) object, in the case of iframe\(^ {362}\), embed\(^ {369}\), or object\(^ {372}\) elements being used to embed SVG.

The `getSVGDocument()` method must run the following steps:

1. Let `document` be this element’s content document\(^ {818}\).
2. If `document` is non-null and was created by the page load processing model for XML files\(^ {486}\) section because the computed type of the resource in the navigate\(^ {487}\) algorithm was image/svg+xml\(^ {5268}\), then return `document`.
3. Return null.
4.8.18 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:

- `specified width - 0.5 ≤ specified height * target ratio ≤ specified width + 0.5`
- `specified height - 0.5 ≤ specified width / target ratio ≤ specified height + 0.5`
- `specified height = 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).

**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.

4.9 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.
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.

Authors are encouraged to provide information describing how to interpret complex tables. Guidance on how to provide such information is given below.

Tables must not be used as layout aids. Historically, some web authors have misused tables in HTML as a way to control their page layout. This usage is non-conforming, because tools attempting to extract tabular data from such documents would obtain very confusing results. In particular, users of accessibility tools like screen readers are likely to find it very difficult to navigate pages with tables used for layout.

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 must not be used as layout aids. Historically, some web authors have misused tables in HTML as a way to control their page layout. This usage is non-conforming, because tools attempting to extract tabular data from such documents would obtain very confusing results. In particular, users of accessibility tools like screen readers are likely to find it very difficult to navigate pages with tables used for layout.

Note

There are a variety of alternatives to using HTML tables for layout, primarily using CSS positioning and the CSS table model.

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 (non-conforming) layout table.

```idl
[Exposed=Window]
interface HTMLTableElement : HTMLElement {
    [HTMLConstructor] constructor();

    [CEReactions] attribute HTMLTableCaptionElement? caption;
    HTMLTableCaptionElement createCaption();
    [CEReactions] undefined deleteCaption();

    [CEReactions] attribute HTMLTableSectionElement? tHead;
    HTMLTableSectionElement createTHead();
    [CEReactions] undefined deleteTHead();

    [CEReactions] attribute HTMLTableSectionElement? tFoot;
    HTMLTableSectionElement createTFoot();
    [CEReactions] undefined deleteTFoot();

    [SameObject] readonly attribute HTMLCollection tBodies;
    HTMLTableSectionElement createTBody();

    [SameObject] readonly attribute HTMLCollection rows;
    HTMLTableRowElement insertRow(optional long index = -1);,
    [CEReactions] undefined deleteRow(long index);

    // also has obsolete members
}
```
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:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Indication</th>
</tr>
</thead>
<tbody>
<tr>
<td>The use of the <strong>role</strong> attribute with the value <code>presentation</code></td>
<td>Probably a layout table</td>
</tr>
<tr>
<td>The use of the non-conforming <strong>border</strong> attribute with the non-conforming value 0</td>
<td>Probably a layout table</td>
</tr>
<tr>
<td>The use of the non-conforming <strong>cellspacing</strong> and <strong>cellpadding</strong> attributes with the value 0</td>
<td>Probably a layout table</td>
</tr>
<tr>
<td>The use of <strong>caption</strong>, <strong>thead</strong>, or <strong>th</strong> elements</td>
<td>Probably a non-layout table</td>
</tr>
<tr>
<td>The use of <strong>headers</strong> and <strong>scope</strong> attributes</td>
<td>Probably a non-layout table</td>
</tr>
<tr>
<td>The use of the non-conforming <strong>border</strong> attribute with a value other than 0</td>
<td>Probably a non-layout table</td>
</tr>
<tr>
<td>Explicit visible borders set using CSS</td>
<td>Probably a non-layout table</td>
</tr>
<tr>
<td>The use of the <strong>summary</strong> attribute</td>
<td>Not a good indicator (both layout and non-layout tables have historically been given this attribute)</td>
</tr>
</tbody>
</table>

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.

For web developers (non-normative)

```
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" DOMException.
```

```
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" DOMException.
```

```
tfoot = table . createTfoot()  
    Ensures the table has a `tfoot` element, and returns it.
```

```
table . deleteTfoot()  
    Ensures the table does not have a `tfoot` element.
```
The `createTBody()` method must return the first `tbody` element child of the `table` element, if any; otherwise a new `tbody` element must be `table-created`, inserted as the first node of the `table` element, and then returned.

The `deleteTBody()` method must remove the first `tbody` element child of the `table` element, if any.

The `createTFoot()` method must return the first `tfoot` element child of the `table` element, if any; otherwise a new `tfoot` element must be `table-created` and inserted immediately before the first element in the `table` element that is neither a `caption` nor a `colgroup` element, if any, or at the end of the table if there are no such elements. If the new value is null, must be inserted as the first node of the `table` element, and then 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 `table-create` a new `tbody` element, insert it immediately after the last `tbody` element child of the `table` element, and then returned.

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" `DOMException`.

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" `DOMException`.

In all of the following attribute and method definitions, when an element is to be **table-created**, that means to create an element given the `table` element's `node document`, the given local name, and the `HTML namespace`.

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 `table-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` nor a `colgroup` element, if any, or at the end of the table if there are no such elements. If the new value is null, must be inserted as the first node of the `table` element, and then returned.

The `createThead()` method must return the first `thead` element child of the `table` element, if any; otherwise a new `thead` element must be `table-created` and inserted immediately before the first element in the `table` element that is neither a `caption` 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" `DOMException` 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 `table-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 `rows` attribute must return an `HTMLCollection` of the `tr` elements of the table.
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 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" `DOMException`.

- **If the `rows` collection has zero elements in it, and the `table` has no `tbody` elements in it:**
  The method must `table-create` a `tbody` element, then `table-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 `table-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 `table-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 `table-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 less than −1 or greater than or equal to the number of elements in the `rows` collection, then throw an "IndexSizeError" `DOMException`.
2. If `index` is −1, then remove the last element in the `rows` collection from its parent, or do nothing if the `rows` collection is empty.
3. Otherwise, remove the `index`th element in the `rows` collection from its parent.

**Example**

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.

```html
<style>
#sudoku { border-collapse: collapse; border: solid thick; }
#sudoku table, #sudoku tbody { border: solid medium; }
#sudoku td { border: solid thin; height: 1.4em; width: 1.4em; text-align: center; padding: 0; }
</style>
<h1>Today's Sudoku</h1>
<table id="sudoku">
  <colgroup><col><col><col></colgroup>
  <tbody>
    <tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr>
    <tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr>
    <tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr>
  </tbody>
</table>
```
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:

<table>
<thead>
<tr>
<th>Negative</th>
<th>Characteristic</th>
<th>Positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sad</td>
<td>Mood</td>
<td>Happy</td>
</tr>
<tr>
<td>Failing</td>
<td>Grade</td>
<td>Passing</td>
</tr>
</tbody>
</table>

...might 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

```xml
<example>
<p>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>
  <caption>Characteristics with positive and negative sides</caption>
  <thead>
    <tr>
      <th id="n">Negative</th>
      <th>Characteristic</th>
      <th>Positive</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td headers="n r1">Sad</td>
      <td id="r1">Mood</td>
      <td>Happy</td>
    </tr>
    <tr>
      <td headers="n r2">Failing</td>
      <td id="r2">Grade</td>
      <td>Passing</td>
    </tr>
  </tbody>
</table>
</example>

In the table's caption
In the table's `<caption>` element, a `<details>` element provides additional information.

**Example**

```html
<table>
  <caption>Characteristics with positive and negative sides.</caption>
  <details>
    <summary>Help</summary>
    <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>
  </details>
  <thead>
    <tr>
      <th id="n">Negative</th>
      <th id="r1">Characteristic</th>
      <th id="r2">Positive</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td headers="n r1">Sad</td>
      <td headers="r1">Mood</td>
      <td headers="r2">Happy</td>
    </tr>
    <tr>
      <td headers="n r2">Failing</td>
      <td headers="r2">Grade</td>
      <td headers="r2">Passing</td>
    </tr>
  </tbody>
</table>
```

Next to the table, in the same `<figure>` element.
Next to the table, in a figure's figcaption.

Authors may also use other techniques, or combinations of the above techniques, as appropriate.

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

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:
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**

- A `caption` element's end tag can be omitted if the `caption` element is not immediately followed by ASCII whitespace or a `comment`.

**Content attributes**

- Global attributes

**Accessibility considerations**

- For authors.
- For implementers.
DOM interface:

IDL:
```javascript
[Exposed=Window]
interface HTMLTableCaptionElement : HTMLElement {
  [HTMLConstructor] constructor();

  // also has obsolete members
};
```

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

Consider, for instance, the following table:

```
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>
```

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:

```
<p>Table 1.</p>
<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.</p>
```

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.
```

4.9.3 The `colgroup` element

Categories:

None.
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 and less than or equal to 1000.

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. It is clamped to the range [1, 1000], and its default value is 1.

### 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` — Number of columns spanned by the element
Accessibility considerations:
For authors.
For implementers.

DOM interface:
Uses `HTMLTableColElement`, as defined for `colgroup` elements.

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 and less than or equal to 1000.

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. It is clamped to the range [1, 1000], and its default value is 1.

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 can 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 can 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

Accessibility considerations:
For authors.
For implementers.

DOM interface:

```
[Exposed=Window]
interface HTMLTableSectionElement : HTMLElement {
  [HTMLConstructor] constructor();

  [SameObject] readonly attribute HTMLCollection rows;
  HTMLTableRowElement insertRow(optional long index = -1);
  [CEReactions] undefined deleteRow(long index);

  // also has obsolete members
};
```

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.

### For web developers (non-normative)

#### `<tbody> . rows`<sup>462</sup>

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" `DOMException`.

#### `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" `DOMException`.

The `rows` attribute must return an `HTMLCollection` rooted at this element, whose filter matches only `<tr>` elements that are children of this element.

The `insertRow(index)` method must act as follows:

1. If `index` is less than −1 or greater than the number of elements in the `rows` collection, throw an "IndexSizeError" `DOMException`.
2. Let `table row` be the result of creating an element given this element's `node document`, `<tr>`, and the HTML namespace.
3. If `index` is −1 or equal to the number of items in the `rows` collection, then append `table row` to this element.
4. Otherwise, insert `table row` as a child of this element, immediately before the `index`th `<tr>` element in the `rows` collection.
5. Return `table row`.

The `deleteRow(index)` method must, when invoked, act as follows:

1. If `index` is less than −1 or greater than or equal to the number of elements in the `rows` collection, then throw an "IndexSizeError" `DOMException`.
2. If `index` is −1, then remove the last element in the `rows` collection from this element, or do nothing if the `rows` collection is empty.
3. Otherwise, remove the `index`th element in the `rows` collection from this element.

### 4.9.6 The `<thead>` element<sup>466</sup>

**Categories**<sup>1129</sup>:

None.

**Contexts in which this element can be used**<sup>1129</sup>:

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**<sup>1129</sup>:

Zero or more `<tr>` and `script-supporting` elements.

**Tag omission in `text/html`**<sup>1130</sup>:

A `<thead>` element’s end tag can be omitted if the `<thead>` element is immediately followed by a `<tbody>` or `<tfoot>` element.
Content attributes:

Global attributes:

Accessibility considerations:

For authors.

For implementers.

DOM interface:

Uses `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

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.

```html
<table>
  <caption>School auction sign-up sheet</caption>
  <thead>
    <tr>
      <th><label for=e1>Name</label></th>
      <th><label for=e2>Product</label></th>
      <th><label for=e3>Picture</label></th>
      <th><label for=e4>Price</label></th>
    </tr>
    <tr>
      <td>Your name here</td>
      <td>What are you selling?</td>
      <td>Link to a picture</td>
      <td>Your reserve price</td>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Ms Danus</td>
      <td>Doughnuts</td>
      <td><img src="https://example.com/mydoughnuts.png" title="Doughnuts from Ms Danus"> $45</td>
    </tr>
    <tr>
      <td><input id=e1 type=text name=who required form=f></td>
      <td><input id=e2 type=text name=what required form=f></td>
      <td><input id=e3 type=url name=pic form=f></td>
      <td><input id=e4 type=number step=0.01 min=0 value=0 required form=f></td>
    </tr>
  </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.
A `<tfoot>` element's end tag can be omitted if there is no more content in the parent element.

### Content attributes

Global attributes

### Accessibility considerations

For authors.

For implementers.

### DOM interface

Uses `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 can 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

### Accessibility considerations

For authors.

For implementers.

### DOM interface

**IDL**

```
[Exposed=Window]
interface HTMLTableRowElement : HTMLElement {
  [HTMLConstructor] constructor();

  readonly attribute long rowIndex;
  readonly attribute long sectionRowIndex;
  [SameObject] readonly attribute HTMLCollection cells;
  HTMLTableCellElement insertCell(optional long index = -1);
  [CEReactions] undefined deleteCell(long index);

  // also has obsolete members
};
```

The `tr` element represents a row of cells in a `table`.

The `tr` element takes part in the `table model`.

---

A `<tfoot>` element's end tag can be omitted if there is no more content in the parent element.

### Content attributes

Global attributes

### Accessibility considerations

For authors.

For implementers.

### DOM interface

Uses `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`.
The `rowIndex` attribute must, if this element has a parent `table` element, or a parent `tbody`, `thead`, or `tfoot` element and a grandparent `table` element, return the index of this `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 this 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 `HTMLTableElement`'s `rows` collection; for table sections, that's `HTMLTableSectionElement`'s `rows` collection). If there is no such parent element, then the attribute must return −1.

The `cells` attribute must return an `HTMLCollection` of the `td` and `th` elements of the row.

`cell = tr.insertCell(index)`

Creates a `td` or `th` 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" `DOMException`.

`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" `DOMException`.

For web developers (non-normative)

The `rowIndex` attribute must, if this element has a parent `table` element, or a parent `tbody`, `thead`, or `tfoot` element and a grandparent `table` element, return the index of this `tr` element in that `table` element's `rows` list. Returns −1 if the element isn't in a table.

The `sectionRowIndex` attribute must, if this element has a parent `table` element, or a parent `tbody`, `thead`, or `tfoot` element, return the index of this `tr` element in that `table` element's `rows` list. Returns −1 if the element isn't in a table section.

The `cells` attribute must return an `HTMLCollection` of the `td` and `th` elements of the row.

The `insertCell(index)` method must act as follows:

1. If `index` is less than −1 or greater than the number of elements in the `cells` collection, then throw an "IndexSizeError" `DOMException`.
2. Let `table cell` be the result of creating an element given this `tr` element's `node document`, `td`, and the `HTML namespace`.
3. If `index` is equal to −1 or equal to the number of items in `cells` collection, then append `table cell` to this `tr` element.
4. Otherwise, insert `table cell` as a child of this `tr` element, immediately before the `index`th `td` or `th` element in the `cells` collection.
5. Return `table cell`.

The `deleteCell(index)` method must act as follows:

1. If `index` is less than −1 or greater than or equal to the number of elements in the `cells` collection, then throw an "IndexSizeError" `DOMException`.
2. If `index` is −1, then remove the last element in the `cells` collection from its parent, or do nothing if the `cells` collection is empty.
3. 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 can 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

Accessibility considerations:
For authors.
For implementers.

DOM interface:

IDL

```javascript
[Exposed=Window]
interface HTMLTableCellElement : HTMLElement {
    [HTMLConstructor] constructor();

    [CEReactions] attribute unsigned long colSpan;
    [CEReactions] attribute unsigned long rowSpan;
    [CEReactions] attribute DOMString headers;
    readonly attribute long cellIndex;

    [CEReactions] attribute DOMString scope; // only conforming for th elements
    [CEReactions] attribute DOMString abbr; // only conforming for th elements

    // also has obsolete members
}
```

The HTMLTableCellElement interface is also used for th elements.

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.

Example

In this example, we see a snippet of a web application consisting of a grid of editable cells (essentially a simple spreadsheet). One of the cells has been configured to show the sum of the cells above it. Three have been marked as headings, which use th elements instead of td elements. A script would attach event handlers to these elements to maintain the total.

```html
<table>
  <tr>
    <th><input value="Name" /></th>
    <th><input value="Paid ($)" /></th>
  </tr>
</table>
```
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 can 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 header cells 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

Accessibility considerations:
For authors.
For implementers.

DOM interface:
Uses HTMLTableCellElement, as defined for td elements.

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 column 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** and **invalid value default** are 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.

**Example**
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:

```html
<table>
<thead>
  <tr>
    <th>ID</th> Measurement <th>Average</th> <th>Maximum</th>
  </tr>
  <tr>
    <td>Cats</td> <th>93</th> <th>Legs</th> 3.5 <th>4</th>
    <td>Tails</td> 1 <th>1</th> 1
  </tr>
  <tr>
    <td>English speakers</td> <th>32</th> <th>Legs</th> 2.67 <th>4</th>
    <td>Tails</td> 0.33 <th>1</th> 1
  </tr>
</tbody>
</table>
```

This would result in the following table:

<table>
<thead>
<tr>
<th>ID</th>
<th>Measurement</th>
<th>Average</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cats</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>Legs</td>
<td>3.5</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>Tails</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>English speakers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Legs</td>
<td>2.67</td>
<td>4</td>
</tr>
<tr>
<td>35</td>
<td>Tails</td>
<td>0.33</td>
<td>1</td>
</tr>
</tbody>
</table>

The headers in the first row all apply directly down to the rows in their column.
The headers with the explicit **scope** attributes apply to all the cells in their row group other than the cells in the first column.
The remaining headers apply just to the cells to the right of them.
The `td` and `th` elements may have a `colspan` content attribute specified, whose value must be a valid non-negative integer greater than zero and less than or equal to 1000.

The `td` and `th` elements may also have a `rowspan` content attribute specified, whose value must be a valid non-negative integer less than or equal to 65534. 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, none of which are identical to another token and 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` 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`. 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.

---

### For web developers (non-normative)

**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. It is clamped to the range \([1, 1000]\), and its default value is 1.

The `rowSpan` IDL attribute must reflect the `rowspan` content attribute. It is clamped to the range \([0, 65534]\), and 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.

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.

### 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 ≤ x < xwidth, and the y coordinates are always in the range 0 ≤ y < yheight. If one or both of xwidth and yheight 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 (cells, celly), and with a particular width and height such that the cell covers all the slots with coordinates (x, y) where `cells ≤ x < cells + width` and `celly ≤ y < celly + 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=xwidth-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=yheight-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, groupy) with a particular height such that the row group covers all the slots with coordinates (x, y) where 0 ≤ x < xwidth and groupy ≤ y < groupy + 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 (groupx, 0) with a particular width such that the column group covers all the slots with coordinates (x, y) where `groupx ≤ x < groupx + width` and 0 ≤ y < yheight. 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 group and zero or more column groups.

In addition to `cells`, `columns`, `rows`, `row groups`, and `column groups`, `table` 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

To determine which elements correspond to which slots in a table associated with a `table` element, to determine the dimensions of the table (xwidth and yheight), and to determine if there are any table model errors, user agents must use the following algorithm:

1. Let `xwidth` be zero.
2. Let `yheight` be zero.
3. Let `pending tfoot` elements be a list of `tfoot` elements, initially empty.
4. Let the `table` be the `table` represented by the `table` element. The `xwidth` and `yheight` variables give the table's dimensions. The `table` is initially empty.
5. If the \texttt{table} element has no children elements, then return the \texttt{table} (which will be empty).

6. Associate the first \texttt{caption} element child of the \texttt{table} element with the \texttt{table}. If there are no such children, then it has no associated \texttt{caption} element.

7. Let the current element be the first element child of the \texttt{table} element.

If a step in this algorithm ever requires the current element to be \texttt{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 \texttt{end}, near the end of this algorithm.

8. While the current element is not one of the following elements, \texttt{advance} the current element to the next child of the \texttt{table}:

- \texttt{colgroup}
- \texttt{thead}
- \texttt{tbody}
- \texttt{tfoot}
- \texttt{tr}

9. If the current element is a \texttt{colgroup}, follow these substeps:

1. Column groups: Process the current element according to the appropriate case below:

   - If the current element has any \texttt{col} element children
     Follow these steps:
     1. Let \texttt{xstart} have the value of \texttt{xwidth}.
     2. Let the current column be the first \texttt{col} element child of the \texttt{colgroup} element.
     3. Columns: If the current column \texttt{col} element has a \texttt{span} attribute, then parse its value using the \texttt{rules for parsing non-negative integers}.
        If the result of parsing the value is not an error or zero, then let \texttt{span} be that value.
        Otherwise, if the \texttt{col} element has no \texttt{span} attribute, or if trying to parse the attribute's value resulted in an error or zero, then let \texttt{span} be 1.
        If \texttt{span} is greater than 1000, let it be 1000 instead.
     4. Increase \texttt{xwidth} by \texttt{span}.
     5. Let the last \texttt{span} columns in the table correspond to the current column \texttt{col} element.
     6. If current column is not the last \texttt{col} element child of the \texttt{colgroup} element, then let the current column be the next \texttt{col} element child of the \texttt{colgroup} element, and return to the step labeled columns.
     7. Let all the last \texttt{span} columns in the table from x=\texttt{xstart} to x=\texttt{xwidth}-1 form a new \texttt{column group} anchored at the slot (\texttt{xstart}, 0), with width \texttt{xwidth-xstart}, corresponding to the \texttt{colgroup} element.

   - If the current element has no \texttt{col} element children
     1. If the \texttt{colgroup} element has a \texttt{span} attribute, then parse its value using the \texttt{rules for parsing non-negative integers}.
        If the result of parsing the value is not an error or zero, then let \texttt{span} be that value.
        Otherwise, if the \texttt{colgroup} element has no \texttt{span} attribute, or if trying to parse the attribute's value resulted in an error or zero, then let \texttt{span} be 1.
        If \texttt{span} is greater than 1000, let it be 1000 instead.
     2. Increase \texttt{xwidth} by \texttt{span}.
     3. Let the last \texttt{span} columns in the table form a new \texttt{column group}, anchored at the slot (\texttt{xwidth-span}, 0), with width \texttt{span}, corresponding to the \texttt{colgroup} element.

2. \texttt{Advance} the current element to the next child of the \texttt{table}.

3. While the current element is not one of the following elements, \texttt{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.

10. Let ycurrent be zero.

11. Let the list of downward-growing cells be an empty list.

12. 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

13. 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.

14. Run the algorithm for ending a row group.

15. If the current element is a tfoot, then add that element to the list of pending tfoot elements, advance the current element to the next child of the table, and return to the step labeled rows.

16. The current element is either a thead or a tbody.

Run the algorithm for processing row groups.

17. Advance the current element to the next child of the table.

18. Return to the step labeled rows.

19. End: For each tfoot element in the list of pending tfoot elements, in tree order, run the algorithm for processing row groups.

20. 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.

21. 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 ystart have the value of yheight.

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 yheight > ystart, then let all the last rows in the table from y=ystart to y=yheight-1 form a new row group, anchored at the slot with coordinate (0, ystart), with height yheight-ystart, 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 ycurrent is less than yheight, follow these steps:

   1. Run the algorithm for growing downward-growing cells.

   2. Increase ycurrent 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 $y_{\text{height}}$ is equal to $y_{\text{current}}$, then increase $y_{\text{height}}$ by 1. ($y_{\text{current}}$ is never greater than $y_{\text{height}}$.)

2. Let $x_{\text{current}}$ be 0.

3. Run the algorithm for growing downward-growing cells\[^{473}\].

4. If the $\text{tr}^{464}$ element being processed has no $\text{td}^{466}$ or $\text{th}^{467}$ element children, then increase $y_{\text{current}}$ by 1, abort this set of steps, and return to the algorithm above.

5. Let $\text{current cell}$ be the first $\text{td}^{466}$ or $\text{th}^{467}$ element child in the $\text{tr}^{464}$ element being processed.

6. **Cells:** While $x_{\text{current}}$ is less than $x_{\text{width}}$ and the slot with coordinate ($x_{\text{current}}, y_{\text{current}}$) already has a cell assigned to it, increase $x_{\text{current}}$ by 1.

7. If $x_{\text{current}}$ is equal to $x_{\text{width}}$, increase $x_{\text{width}}$ by 1. ($x_{\text{current}}$ is never greater than $x_{\text{width}}$.)

8. If the $\text{current cell}$ has a $\text{colspan}^{466}$ attribute, then parse that attribute's value\[^{469}\], 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.

   If colspan is greater than 1000, let it be 1000 instead.

9. If the $\text{current cell}$ has a $\text{rowspan}^{469}$ attribute, then parse that attribute's value\[^{469}\], and let rowspan be the result.

   If parsing that value failed or if the attribute is absent, then let rowspan be 1, instead.

   If rowspan is greater than 65534, let it be 65534 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 $x_{\text{width}} < x_{\text{current}} + \text{colspan}$, then let $x_{\text{width}}$ be $x_{\text{current}} + \text{colspan}$.

12. If $y_{\text{height}} < y_{\text{current}} + \text{rowspan}$, then let $y_{\text{height}}$ be $y_{\text{current}} + \text{rowspan}$.

13. Let the slots with coordinates ($x, y$) such that $x_{\text{current}} \leq x < x_{\text{current}} + \text{colspan}$ and $y_{\text{current}} \leq y < y_{\text{current}} + \text{rowspan}$ be covered by a new $\text{cell}^{470} c$, anchored at ($x_{\text{current}}, y_{\text{current}}$), which has width colspan and height rowspan, corresponding to the current cell element.

   If the current cell element is a $\text{th}^{467}$ 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\[^{473}\], described in the next section.

   If any of the slots involved already had a $\text{cell}^{470}$ covering them, then this is a table model error\[^{470}\]. Those slots now have two cells overlapping.

14. If cell grows downward is true, then add the tuple { $c, x_{\text{current}}, \text{colspan}$} to the list of downward-growing cells.

15. Increase $x_{\text{current}}$ by $\text{colspan}$.

16. If $\text{current cell}$ is the last $\text{td}^{466}$ or $\text{th}^{467}$ element child in the $\text{tr}^{464}$ element being processed, then increase $y_{\text{current}}$ by 1, abort this set of steps, and return to the algorithm above.

17. Let $\text{current cell}$ be the next $\text{td}^{466}$ or $\text{th}^{467}$ element child in the $\text{tr}^{464}$ 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, $x_{\text{cell}}, \text{width}$} tuple in the list of downward-growing cells, if any, extend the cell\[^{470}\] cell so that it also covers the slots with coordinates ($x, y_{\text{current}}$), where $x_{\text{cell}} \leq x < x_{\text{cell}} + \text{width}$.

### 4.9.12.2 Forming relationships between data cells and header cells \[^{473}\]

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 ($\text{principal}_x, \text{principal}_y$) 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 ASCII whitespace, 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.

4. 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 Δx=−1 and Δy=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 Δx=0 and Δy=−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 principal x+principal width-1 and a y-coordinate less than or equal to principal y+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 coordinate (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 are 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.
   5. 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.
   6. If blocked is false, then add the current cell to the headers list.

   4. 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 .. y+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 .. x+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 child text content, if any, consists only of ASCII whitespace.

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:

```xml
<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>
    </tr>
  </thead>
</table>
```
<table>
<thead>
<tr>
<th>Grade</th>
<th>Yield Point</th>
<th>Ultimate tensile strength</th>
<th>Per cent elong. 50.8mm or 2 in.</th>
<th>Per cent reduct. area.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>kg/mm²</td>
<td>lb/in²</td>
<td></td>
</tr>
<tr>
<td>Hard</td>
<td>0.45 ultimate</td>
<td>56.2</td>
<td>80,000</td>
<td>15</td>
</tr>
<tr>
<td>Medium</td>
<td>0.45 ultimate</td>
<td>49.2</td>
<td>70,000</td>
<td>18</td>
</tr>
<tr>
<td>Soft</td>
<td>0.45 ultimate</td>
<td>42.2</td>
<td>60,000</td>
<td>22</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.

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:
The following shows how one might mark up the operating expenses table from lower on the same page of that document:

```
<table>
  <colgroup><col><col><col></colgroup>
  <thead>
    <tr><th scope=rowgroup>Research and development</th>
        <td>$ 1,109</td><td>$ 782</td><td>$ 712</td></tr>
    <tr><th scope=row>Percentage of net sales</th>
        <td>3.4%</td><td>3.3%</td><td>3.7%</td></tr>
  </thead>
  <tbody>
    <tr><th scope=rowgroup>Selling, general, and administrative</th>
        <td>$ 3,761</td><td>$ 2,963</td><td>$ 2,433</td></tr>
    <tr><th scope=row>Percentage of net sales</th>
        <td>11.6%</td><td>12.3%</td><td>12.6%</td></tr>
  </tbody>
</table>
```

This table could look like this:

<table>
<thead>
<tr>
<th></th>
<th>2008</th>
<th>2007</th>
<th>2006</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net sales</td>
<td>$32,479</td>
<td>$24,006</td>
<td>$19,315</td>
</tr>
<tr>
<td>Cost of sales</td>
<td>21,334</td>
<td>15,852</td>
<td>13,717</td>
</tr>
<tr>
<td>Gross margin</td>
<td>$11,145</td>
<td>$8,154</td>
<td>$5,598</td>
</tr>
<tr>
<td>Gross margin percentage</td>
<td>34.3%</td>
<td>34.0%</td>
<td>29.0%</td>
</tr>
</tbody>
</table>
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, buttons, checkboxes, range, or color picker controls. 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 text control. To label a control, the `label` element is used; the label text and the control itself go inside the `label` element. Each part of a form is considered a paragraph, and is typically separated from other parts using `p` elements. Putting this together, here is how one might ask for the customer's name:

```html
<form>
  <p><label>Customer name: <input></label></p>
</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.

```html
<form>
  <p><label>Customer name: <input></label></p>
  <fieldset>
    <legend>Pizza Size</legend>
    <p><label><input type=radio name=size> Small</label></p>
    <p><label><input type=radio name=size> Medium</label></p>
    <p><label><input type=radio name=size> Large</label></p>
  </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`:

```html
<form>
  <p><label>Customer name: <input></label></p>
  <fieldset>
    <legend>Pizza Size</legend>
    <p><label><input type=radio name=size> Small</label></p>
    <p><label><input type=radio name=size> Medium</label></p>
    <p><label><input type=radio name=size> Large</label></p>
  </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 email addresses (input elements with their type attribute set to email):

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:

The textarea element can be used to provide a multiline text control. In this instance, we are going to use it to provide a space for the customer to give delivery instructions:
Finally, to make the form submittable we use the button element:

```
<form>
  <p><label>Customer name: <input/></label></p>
  <p><label>Telephone: <input type=tel/></label></p>
  <p><label>Email address: <input type=email/></label></p>
  <fieldset>
    <legend>Pizza Size</legend>
    <p><label><input type=radio name=size> Small </label></p>
    <p><label><input type=radio name=size> Medium </label></p>
    <p><label><input type=radio name=size> Large </label></p>
  </fieldset>
  <fieldset>
    <legend>Pizza Toppings</legend>
    <p><label><input type=checkbox> Bacon </label></p>
    <p><label><input type=checkbox> Extra Cheese </label></p>
    <p><label><input type=checkbox> Onion </label></p>
    <p><label><input type=checkbox> Mushroom </label></p>
  </fieldset>
  <p><label>Preferred delivery time: <input type=time min="11:00" max="21:00" step="900"></label></p>
  <p><label>Delivery instructions: <textarea></textarea></label></p>
  <button>Submit order</button>
</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:

- **username**
  - Customer's name
- **custtel**
  - Customer's telephone number
- **custemail**
  - Customer's email 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:

```html
<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>Email 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" 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 the syntax section.

For example, if the customer entered "Denise Lawrence" as their name, "555-321-8642" as their telephone number, did not specify an
email 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 control 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

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:

```html
<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>Email 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>
  <button>Submit order</button>
</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:

```html
<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>Email 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>
```
<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>Email 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>
  Preferred delivery time: <input type="time" min="11:00" max="21:00" step="900" name="delivery" required/>
  <label>Delivery instructions: <textarea name="comments" maxlength=1000></textarea></label>
  <button>Submit order</button>
</form>

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:

```html
<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>Email 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>
  Preferred delivery time: <input type="time" min="11:00" max="21:00" step="900" name="delivery" required/>
  <label>Delivery instructions: <textarea name="comments" maxlength=1000></textarea></label>
  <button>Submit order</button>
</form>

4.10.1.6 Improving the user experience on mobile devices

This section is non-normative.

Some devices, in particular those with virtual keyboards 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:

```html
<form method="post" enctype="application/x-www-form-urlencoded"
  action="https://pizza.example.com/order.cgi">
  <p><label for="customer name">Customer name: <input name="custname" required autocomplete="shipping name"></label></p>
  <p><label for="telephone">Telephone: <input type="tel" name="custtel" autocomplete="shipping tel"></label></p>
  <p><label for="buzzer code">Buzzer code: <input name="custbuzz" inputmode="numeric"></label></p>
  <p><label for="email address">Email address: <input type="email" name="customail" autocomplete="shipping email"></label></p>
  <fieldset>
    <legend>Pizza Size</legend>
    <input type="radio" name="size required value="small" required value="small"> Small</input>
    <input type="radio" name="size required value="medium" required value="medium"> Medium</input>
    <input type="radio" name="size required value="large" required value="large"> Large
  </fieldset>
  <fieldset>
    <legend>Pizza Toppings</legend>
    <input type="checkbox" name="topping" value="bacon"> Bacon<input>
    <input type="checkbox" name="topping" value="cheese"> Extra Cheese<input>
    <input type="checkbox" name="topping" value="onion"> Onion<input>
    <input type="checkbox" name="topping" value="mushroom"> Mushroom<input>
  </fieldset>
  <p>PREFERRED delivery time: <input type="time" min="11:00" max="21:00" step="900" name="delivery required"></input>
  <p>Delivery instructions: <textarea name="comments" maxlength="1000"></textarea>
  <button>Submit order</button>
</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 `<input>` attribute, `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 `<input>` 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, 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:

```html
<%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`):

```html
"https://pizza.example.com/order.cgi"?
```
Your phone number: <input type=tel name=custtel autocomplete="billing tel-national"/>
Recipient's phone number: <input type=tel name=shiptel autocomplete="shipping tel-national"/>

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 text control (<input type=text>), a drop-down list (<select>), radio buttons (<input type=radio>), etc. It only depends on what kind of interface is desired.

Finally, 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.

<p><label>Japanese name: <input name=j type=text autocomplete="section-jp name"></label></p>
<p><label>Romanized name: <input name=e type=text autocomplete="section-en name"></label></p>

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.
</Note>

Separate from the choices regarding type and autocomplete, the inputmode attribute decides what kind of input modality (e.g., virtual keyboard) to use, when the control is a text control.

Consider credit card numbers. The appropriate input type is not <input type=number>, as explained below; it is instead <input type=text>. To encourage the user agent to use a numeric input modality anyway (e.g., a virtual keyboard displaying only digits), the page would use

<p><label>Credit card number: <input name="cc" type=text inputmode="numeric" pattern="[0-9]{8,19}" autocomplete="cc-number"></label></p>

### 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 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" — and even with numbers, where locales differ, for example, in what punctuation is used as the decimal separator and the thousands separator.

It is therefore important to distinguish the time, date, and number formats used in HTML and in form submissions, which are always the formats defined in this specification (and based on the well-established ISO 8601 standard for computer-readable date and time formats), from the time, date, and number formats presented to the user by the browser and accepted as input from the user by the browser.

The format used "on the wire", i.e., in HTML markup and in form submissions, is intended to be computer-readable and consistent irrespective of the user's locale. Dates, for instance, are always written in the format "YYYY-MM-DD", as in "2003-02-01". While some users might see this format, others might see it as "01.02.2003" or "February 1, 2003".
The time, date, or number given by the page in the wire format is then translated to the user's preferred presentation (based on user preferences or on the locale of the page itself), before being displayed to the user. Similarly, after the user inputs a time, date, or number using their preferred format, the user agent 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`, `object`, `output`, `select`, `textarea`, `img`, form-associated custom elements

The form-associated elements fall into several subcategories:

**Listed elements**

Denotes elements that are listed in the `form.elements` and `fieldset.elements` APIs. These elements also have a form content attribute, and a matching form IDL attribute, that allow authors to specify an explicit form owner.

- `button`, `fieldset`, `input`, `object`, `output`, `select`, `textarea`, form-associated custom elements

**Submittable elements**

Denotes elements that can be used for constructing the entry list when a form element is submitted.

- `button`, `input`, `select`, `textarea`, form-associated custom elements

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`, `output`, `select`, `textarea`, form-associated custom elements

**Autocapitalize-inheriting elements**

Denotes elements that inherit the autocapitalize attribute from their form owner.

- `button`, `fieldset`, `input`, `object`, `output`, `select`, `textarea`, form-associated custom elements

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), `meter`, `output`, `progress`, `select`, `textarea`, form-associated custom elements

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 but with no element descendants.

Tag omission in text/html:
Neither tag is omissible.

Content attributes:
- Global attributes:
  - accept-charset — Character encodings to use for form submission.
  - action — URL to use for form submission.
  - autocomplete — Default setting for autofill feature for controls in the form.
  - enctype — Entry list encoding type to use for form submission.
  - method — Variant to use for form submission.
  - name — Name of form to use in the document.forms API.
  - novalidate — Bypass form control validation for form submission.
  - target — Browsing context for form submission.
  - rel —

Accessibility considerations:
- For authors.
- For implementers.

DOM interface:

```
[Exposed=Window,
 LegacyOverrideBuiltIns,
 LegacyUnenumerableNamedProperties]
interface HTMLFormElement : HTMLElement {
    [HTMLConstructor] constructor();

    [CEReactions] attribute DOMString acceptCharset;
    [CEReactions] attribute USVString action;
    [CEReactions] attribute DOMString autocomplete;
    [CEReactions] attribute DOMString enctype;
    [CEReactions] attribute DOMString encoding;
    [CEReactions] attribute DOMString method;
    [CEReactions] attribute DOMString name;
    [CEReactions] attribute boolean noValidate;
    [CEReactions] attribute DOMString target;
    [CEReactions] attribute DOMString rel;
    [SameObject, PutForwards=value] readonly attribute DOMTokenList relList;

    [SameObject] readonly attribute HTMLFormControlsCollection elements;
    readonly attribute unsigned long length;
    getter Element (unsigned long index);
    getter (RadioNodeList or Element) (DOMString name);

    undefined submit();
    undefined requestSubmit(optional HTMLElement? submitter = null);
    [CEReactions] undefined reset();
    boolean checkValidity();
    boolean reportValidity();
};
```

The form element represents a hyperlink that can be manipulated through 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 attribute gives the character encodings that are to be used for the submission. If specified, the value must be an ASCII case-insensitive match for "UTF-8". [ENCODING]

The name 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` 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 and the invalid value default are the `on` state. The `off` state indicates that by default, form controls in the form will have their `autofill field name` set to "on". 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`, `novalidate`, and `target` attributes are attributes for form submission.

The `rel` attribute on `form` elements controls what kinds of links the elements create. The attribute's value must be an unordered set of unique space-separated tokens. The allowed keywords and their meanings are defined in an earlier section.

`rel`'s supported tokens are the keywords defined in HTML link types, which are allowed on `form` elements, impact the processing model, and are supported by the user agent. The possible supported tokens are `noreferrer`, `noopener`, and `opener`. `rel`'s supported tokens must only include the tokens from this list that the user agent implements the processing model for.

For web developers (non-normative)

```javascript
form.elements
```

Returns an `HTMLFormControlsCollection` of the form controls in the form (excluding image buttons for historical reasons).

```javascript
form.length
```

Returns the number of form controls in the form (excluding image buttons for historical reasons).

```javascript
form[index]
```

Returns the indexth element in the form (excluding image buttons for historical reasons).

```javascript
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 tree. If there are multiple matching items, then a `RadioNodeList` object containing all those elements is returned.

```javascript
form.submit()
```

Submits the form, bypassing interactive constraint validation, and without firing a `submit` event.

```javascript
form.requestSubmit([submitter])
```

Requests to submit the form. Unlike `submit()`, this method includes interactive constraint validation and firing a `submit` event, either of which can cancel submission.

The `submitter` argument can be used to point to a specific `submit button` whose `formaction`, `formmethod`, `formenctype`, `formmethod`, `formnovalidate`, and `formtarget` attributes can impact submission. Additionally, the submitter will be included when constructing the entry list for submission; normally, buttons are excluded.

```javascript
form.reset()
```

Resets the form.

```javascript
form.checkValidity()
```

Returns true if the form's controls are all valid; otherwise, returns false.

```javascript
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` and `rel` IDL attributes must reflect the content attribute of the same name.

The `acceptCharset` IDL attribute must reflect the `accept-charset` content attribute.

The `relist` IDL attribute must reflect the `rel` content attribute.

The `elements` IDL attribute must return an `HTMLFormControlsCollection` rooted at the `form` element's `root`, 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.

To determine the value of an indexed property for a form element, 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:

1. 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.
2. 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:
   1. 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.
   2. 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.
3. For each img element candidate whose form owner is the form element:
   1. 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.
   2. 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.
4. 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.
5. 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.
6. Remove any entries in sourced names that have the empty string as their name.
7. Remove any entries in sourced names that have the same name as an earlier entry in the map.
8. Return the list of names from sourced names, maintaining their relative order.

To determine the value of a named property name for a form element, the user agent must run the following steps:

1. 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.
2. If candidates is empty, let candidates be a live RadioNodeList object containing all the img elements, whose form owner is the form element, that have either an id attribute or a name attribute equal to name, in tree order.
3. 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.
4. If candidates contains more than one node, return candidates.
5. 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.
6. 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
The submit() method, when invoked, must submit the form element from the form element itself, with the submitted from submit() method flag set.

The requestSubmit(submitter) method, when invoked, must run the following steps:

1. If submitter is not null, then:
   1. If submitter is not a submit button, then throw a TypeError.
   2. If submitter's form owner is not this form element, then throw a "NotFoundError" DOMException.
2. Otherwise, set submitter to this form element.
3. Submit this form element, from submitter.

The reset() method, when invoked, must run the following steps:

1. If the form element is marked as locked for reset, then return.
2. Mark the form element as locked for reset.
3. Reset the form element.
4. 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

This example shows two search forms:

```html
<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
- 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.
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 tree as the `label` element. If the attribute is specified and there is an element in the tree whose ID is equal to the value of the `for` attribute, and the first such element in tree order is a labelable element, then that element is the `label` element's labeled control.

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 exact default presentation and behavior, in particular what its activation behavior might be, if anything, should match the platform's label behavior. The activation behavior of a `label` element for events targeted at interactive descendants of a `label` element, and any descendants of those interactive descendants, must be to do nothing.

Form-associated custom elements are labelable elements, so for user agents where the `label` element's activation behavior impacts the labeled control, both built-in and custom elements will be impacted.

Example

For example, on platforms where clicking a label activates the form control, clicking the `label` in the following snippet could trigger the user agent to fire a click event at the `input` element, as if the element itself had been triggered by the user:

```html
<label><input type=checkbox name=lost> Lost</label>
```

Similarly, assuming `my-checkbox` was declared as a form-associated custom element (like in this example), then the code

```html
<label><my-checkbox name=lost></my-checkbox> Lost</label>
```

would have the same behavior, firing a click event at the `my-checkbox` element.

On other platforms, the behavior in both cases might be just to focus the control, or to do nothing.

Example

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.
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.

The `form` IDL attribute must run the following steps:

1. If the `label` element has no `labeled control`, then return null.
2. If the `label` element’s `labeled control` is not a `form-associated element`, then return null.
3. Return the `label` element’s `labeled control`’s `form owner` (which can still be null).

The `htmlFor` IDL attribute on the `label` element is different from the `form` IDL attribute on listed `form-associated elements`, and the `label` element does not have a `form` content attribute.

Note

The `form` IDL attribute on the `label` element is different from the `form` IDL attribute on listed `form-associated elements`, and the `label` element does not have a `form` content attribute.

For web developers (non-normative)

- `label.control` Returns the form control that is associated with this element.
- `label.form` Returns the `form owner` of the form control that is associated with this element. Returns null if there isn’t one.

Labelable elements and all `input` elements have a `live` `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` that are not `form-associated custom elements`, and the `labels` IDL attribute of `input` elements, on getting, must return that `NodeList` object, and that same value must always be returned, unless this element is an `input` element whose `type` attribute is in the `Hidden` state, in which case it must instead return null.

Form-associated custom elements don’t have a `labels` IDL attribute. Instead, their `ElementInternals` object has a `labels` IDL attribute. On getting, it must throw a `NotFoundError` DOMException if the target element is not a form-associated custom element. Otherwise, it must return that `NodeList` object, and that same value must always be returned.

Example

This (non-conforming) example shows what happens to the `NodeList` and what `labels` returns when an `input` element has its `type` attribute changed.
input.type = 'checkbox';
console.assert(labels.length === 1); // the input is once again the label's labeled control
console.assert(input.labels === labels); // same value as returned originally
</script>

4.10.5 The `input` element

Categories:
- Flow content
- Phrasing content

If the `type` attribute is not in the `Hidden` state:
- Interactive content
  - Listed, labelable, submittable, resettable, and autocapitalize-inheriting form-associated element.
If the `type` attribute is in the `Hidden` state:
- Listed, submittable, resettable, and autocapitalize-inheriting 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
  - checked — Whether the control is checked
  - dirname — Name of form control to use for sending the element’s directionality in form submission
  - disabled — Whether the form control is disabled
  - form — Associates the element with a form element
  - formaction — URL to use for form submission
  - formenctype — Entry list encoding type to use for form submission
  - formmethod — Variant to use for form submission
  - formnovalidate — Bypass form control validation for form submission
  - formtarget — Browsing context for form submission
  - height — Vertical dimension
  - 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 the element to use for 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 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
Accessibility considerations:

- `type` attribute in the `Hidden` state: for authors; for implementers.
- `type` attribute in the `Text` state: for authors; for implementers.
- `type` attribute in the `Search` state: for authors; for implementers.
- `type` attribute in the `Telephone` state: for authors; for implementers.
- `type` attribute in the `Image Button` state: for authors; for implementers.
- `type` attribute in the `Color` state: for authors; for implementers.
- `type` attribute in the `Password` state: for authors; for implementers.
- `type` attribute in the `FormTarget` state: for authors; for implementers.
- `type` attribute in the `FormMethod` state: for authors; for implementers.
- `type` attribute in the `Attendees` state: for authors; for implementers.
- `type` attribute in the `Button` state: for authors; for implementers.

DOM interface:

```javascript
[Exposed=Window]
interface HTMLInputElement : HTMLElement {
    [HTMLConstructor] constructor();

    [CEReactions] attribute DOMString accept;
    [CEReactions] attribute DOMString alt;
    [CEReactions] attribute boolean defaultChecked;
    attribute boolean checked;
    [CEReactions] attribute DOMString dirName;
    [CEReactions] attribute boolean disabled;
    readonly attribute HTMLFormElement? form;
    attribute FileList? files;
    [CEReactions] attribute USVString formAction;
    [CEReactions] attribute DOMString formEnctype;
    [CEReactions] attribute DOMString formMethod;
    [CEReactions] attribute boolean formNoValidate;
    [CEReactions] attribute DOMString formTarget;
    [CEReactions] attribute unsigned long height;
    attribute boolean indeterminate;
    readonly attribute HTMLInputElement? list;
    [CEReactions] attribute DOMString max;
    [CEReactions] attribute long maxLen;
    [CEReactions] attribute DOMString min;
    [CEReactions] attribute long minLength;
    [CEReactions] attribute boolean multiple;
    [CEReactions] attribute DOMString name;
    [CEReactions] attribute DOMString pattern;
    [CEReactions] attribute DOMString placeholder;
    [CEReactions] attribute boolean readOnly;
    [CEReactions] attribute boolean required;
    [CEReactions] attribute unsigned long size;
```

Also, the `title` attribute has special semantics on this element: Description of pattern (when used with `pattern`).
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 (and associated control) of the element. 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.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>State</th>
<th>Data type</th>
<th>Control type</th>
</tr>
</thead>
<tbody>
<tr>
<td>hidden</td>
<td>Hidden</td>
<td>An arbitrary string</td>
<td>n/a</td>
</tr>
<tr>
<td>text</td>
<td>Text</td>
<td>Text with no line breaks</td>
<td>A text control</td>
</tr>
<tr>
<td>search</td>
<td>Search</td>
<td>Text with no line breaks</td>
<td>Search control</td>
</tr>
<tr>
<td>tel</td>
<td>Telephone</td>
<td>Text with no line breaks</td>
<td>A text control</td>
</tr>
<tr>
<td>url</td>
<td>URL</td>
<td>An absolute URL</td>
<td>A text control</td>
</tr>
<tr>
<td>email</td>
<td>Email</td>
<td>An email address or list of email addresses</td>
<td>A text control</td>
</tr>
<tr>
<td>password</td>
<td>Password</td>
<td>Text with no line breaks (sensitive information)</td>
<td>A text control that obscures data entry</td>
</tr>
<tr>
<td>date</td>
<td>Date</td>
<td>A date (year, month, day) with no time zone</td>
<td>A date control</td>
</tr>
<tr>
<td>month</td>
<td>Month</td>
<td>A date consisting of a year and a month with no time zone</td>
<td>A month control</td>
</tr>
<tr>
<td>week</td>
<td>Week</td>
<td>A date consisting of a week-year number and a week number with no time zone</td>
<td>A week control</td>
</tr>
<tr>
<td>time</td>
<td>Time</td>
<td>A time (hour, minute, seconds, fractional seconds) with no time zone</td>
<td>A time control</td>
</tr>
<tr>
<td>datetimelocal</td>
<td>Local Date and Time</td>
<td>A date and time (year, month, day, hour, minute, second, fraction of a second) with no time zone</td>
<td>A date and time control</td>
</tr>
<tr>
<td>number</td>
<td>Number</td>
<td>A numerical value</td>
<td>A text control or spinner control</td>
</tr>
<tr>
<td>range</td>
<td>Range</td>
<td>A numerical value, with the extra semantic that the exact value is not important</td>
<td>A slider control or similar</td>
</tr>
<tr>
<td>color</td>
<td>Color</td>
<td>An sRGB color with 8-bit red, green, and blue components</td>
<td>A color picker</td>
</tr>
</tbody>
</table>
The missing value default and the invalid value default are the Text state.

Which of the accept, alt, autocomplete, checked, dirname, formaction, formenctype, formmethod, formnovalidate, formtarget, height, list, max, maxlength, min, minlength, multiple, pattern, placeholder, size, src, step, and width content attributes, the checkbox, 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:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>State</th>
<th>Data type</th>
<th>Control type</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkbox</td>
<td>Checked</td>
<td>A set of zero or more values from a predefined list</td>
<td>A checkbox</td>
</tr>
<tr>
<td>radio</td>
<td>Radio Button</td>
<td>An enumerated value</td>
<td>A radio button</td>
</tr>
<tr>
<td>file</td>
<td>File Upload</td>
<td>Zero or more files each with a MIME type and optionally a file name</td>
<td>A label and a button</td>
</tr>
<tr>
<td>submit</td>
<td>Submit Button</td>
<td>An enumerated value, with the extra semantic that it must be the last value selected and initiates form submission</td>
<td>A button</td>
</tr>
<tr>
<td>image</td>
<td>Image Button</td>
<td>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</td>
<td>Either a clickable image, or a button</td>
</tr>
<tr>
<td>reset</td>
<td>Reset Button</td>
<td>n/a</td>
<td>A button</td>
</tr>
<tr>
<td>button</td>
<td>Button</td>
<td>n/a</td>
<td>A button</td>
</tr>
</tbody>
</table>

The missing value default and the invalid value default are the Text state.

Which of the accept, alt, autocomplete, checked, dirname, formaction, formenctype, formmethod, formnovalidate, formtarget, height, list, max, maxlength, min, minlength, multiple, pattern, placeholder, size, src, step, and width content attributes, the checkbox, 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:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>State</th>
<th>Data type</th>
<th>Control type</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkbox</td>
<td>Checked</td>
<td>A set of zero or more values from a predefined list</td>
<td>A checkbox</td>
</tr>
<tr>
<td>radio</td>
<td>Radio Button</td>
<td>An enumerated value</td>
<td>A radio button</td>
</tr>
<tr>
<td>file</td>
<td>File Upload</td>
<td>Zero or more files each with a MIME type and optionally a file name</td>
<td>A label and a button</td>
</tr>
<tr>
<td>submit</td>
<td>Submit Button</td>
<td>An enumerated value, with the extra semantic that it must be the last value selected and initiates form submission</td>
<td>A button</td>
</tr>
<tr>
<td>image</td>
<td>Image Button</td>
<td>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</td>
<td>Either a clickable image, or a button</td>
</tr>
<tr>
<td>reset</td>
<td>Reset Button</td>
<td>n/a</td>
<td>A button</td>
</tr>
<tr>
<td>button</td>
<td>Button</td>
<td>n/a</td>
<td>A button</td>
</tr>
</tbody>
</table>

The missing value default and the invalid value default are the Text state.

Which of the accept, alt, autocomplete, checked, dirname, formaction, formenctype, formmethod, formnovalidate, formtarget, height, list, max, maxlength, min, minlength, multiple, pattern, placeholder, size, src, step, and width content attributes, the checkbox, 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 summaries which of those content attributes, IDL attributes, and methods and events apply to each state:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>State</th>
<th>Data type</th>
<th>Control type</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkbox</td>
<td>Checked</td>
<td>A set of zero or more values from a predefined list</td>
<td>A checkbox</td>
</tr>
<tr>
<td>radio</td>
<td>Radio Button</td>
<td>An enumerated value</td>
<td>A radio button</td>
</tr>
<tr>
<td>file</td>
<td>File Upload</td>
<td>Zero or more files each with a MIME type and optionally a file name</td>
<td>A label and a button</td>
</tr>
<tr>
<td>submit</td>
<td>Submit Button</td>
<td>An enumerated value, with the extra semantic that it must be the last value selected and initiates form submission</td>
<td>A button</td>
</tr>
<tr>
<td>image</td>
<td>Image Button</td>
<td>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</td>
<td>Either a clickable image, or a button</td>
</tr>
<tr>
<td>reset</td>
<td>Reset Button</td>
<td>n/a</td>
<td>A button</td>
</tr>
<tr>
<td>button</td>
<td>Button</td>
<td>n/a</td>
<td>A button</td>
</tr>
</tbody>
</table>
Some states of the `type` attribute define a value sanitization algorithm.

Each `input` element has a `value` attribute, 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`, `valueAsDate`, `valueAsNumber`, and `stepUp()`.

An `input` element’s `dirty value flag` 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`. 

---

<table>
<thead>
<tr>
<th>Hidden</th>
<th>Text Search</th>
<th>URL Telephone</th>
<th>Email</th>
<th>Password</th>
<th>Date Month</th>
<th>Week</th>
<th>Time</th>
<th>Local Date Time</th>
<th>Number</th>
<th>Range</th>
<th>Color</th>
<th>Checkbox</th>
<th>RadioButton</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>value</code></td>
<td><code>value</code></td>
<td><code>value</code></td>
<td><code>value</code></td>
<td><code>value</code></td>
<td><code>value</code></td>
<td><code>value</code></td>
<td><code>value</code></td>
<td><code>value</code></td>
<td><code>value</code></td>
<td><code>value</code></td>
<td><code>value</code></td>
<td><code>value</code></td>
<td><code>default</code></td>
</tr>
<tr>
<td><code>valueAsDate</code></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td><code>valueAsNumber</code></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td><code>list</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><code>select()</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td><code>selectionStart</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><code>selectionEnd</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><code>setRangeText()</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><code>setSelectionRange()</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><code>stepDown()</code></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td><code>stepUp()</code></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
</tbody>
</table>

† If the control has no selectable text, the `select()` method results in a no-op, with no `InvalidStateError` DOMException.
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.

The activation behavior for input elements are these steps:

1. If this element is not mutable and is not in the Checkbox state and is not in the Radio state, then return.

2. Run this element's input activation behavior, if any, and do nothing otherwise.

The legacy-pre-activation behavior for input elements are these steps:

1. If this element's type attribute is in the Checkbox state, then set this element's checkedness to its opposite value (i.e. true if it is false, false if it is true) and set this element's indeterminate IDL attribute to false.

2. If this element's type attribute is in the Radio Button state, then get a reference to the element in this element's radio button group that has its checkedness set to true, if any, and then set this element's checkedness to true.

The legacy-cancelled-activation behavior for input elements are these steps:

1. If the element's type attribute is in the Checkbox state, then set the element's checkedness and the element's indeterminate IDL attribute to the values they had before the legacy-pre-activation behavior was run.

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 put 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.

7. Let previouslySelectable be true if setRangeText() previously applied to the element, and false otherwise.

8. Let nowSelectable be true if setRangeText() now applies to the element, and false otherwise.

9. If previouslySelectable is false and nowSelectable is true, set the element's text entry cursor position to the beginning of the text control, and set its selection direction to "none".

The name attribute represents the element's name. The disabled attribute controls how the element's directionality is submitted. The dirname 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. 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 defaultValue IDL attribute must reflect the value content attribute. The checked content attribute. The defaultChecked 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 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][1270]

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's. The select(), selectionStart, selectionEnd, selectionDirection, setRangeText(), and setSelectionRange() methods and IDL attributes expose the element's text selection. The 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)

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 an ASCII case-insensitive match for "_charset_", then the element's value attribute must be omitted.

**Bookkeeping details**

- The autocomplete content attribute applies to this element.
- 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, checked, dirName, formaction, formenctype, formnovalidate, formtarget, formtype, formmethod, height, 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)

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 controls are distinguished from regular text controls, the Search state might result in an appearance consistent with the platform's search controls.
controls rather than appearing like a regular text control.

If the element is `mutable`, its `value` should be editable by the user. User agents must not allow users to insert U+00A LINE FEED (LF) or U+00D 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 an element task on the user interaction task source given the element to fire an event named `input` at the element, with the `bubbles` and `composed` attributes initialized to true.

The `value`, if specified, must have a value that contains no U+00A LINE FEED (LF) or U+00D CARRIAGE RETURN (CR) characters.

The `value sanitization algorithm` is as follows: Strip newlines from the `value`.

**Bookkeeping details**
- The following common `input` element content attributes, IDL attributes, and methods `apply` to the element: `autocomplete`, `dir`, `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 common `input` and `change` events `apply`.

**4.10.5.1.3 Telephone state (type=tel)**

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+00A LINE FEED (LF) or U+00D CARRIAGE RETURN (CR) characters into the element's `value`.

The `value`, if specified, must have a value that contains no U+00A LINE FEED (LF) or U+00D CARRIAGE RETURN (CR) characters.

The `value sanitization algorithm` is as follows: Strip newlines from the `value`.

**Note**

Unlike the `URL` and `Email` 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`, `dir`, `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`, `dir`, `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.
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`, 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 newlines from the `value`, then strip leading and trailing ASCII 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`, `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

If a document contained the following markup:

```html
<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" value="https://html.spec.whatwg.org/">
  <option label="DOM" value="https://dom.spec.whatwg.org/">
  <option label="Fullscreen" value="https://fullscreen.spec.whatwg.org/">
  <option label="Media Session" value="https://mediasection.spec.whatwg.org/">
</datalist>
```

...and the user had typed "spec.w", and the user agent had also found that the user had visited https://url.spec.whatwg.org/#url-parsing and https://streams.spec.whatwg.org/ 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,
When an input element’s type attribute is in the Email state, the rules in this section apply.

How the Email 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 email address given in the element’s value.

If the element is mutable, the user agent should allow the user to change the email address represented by its value. User agents may allow the user to set the value to a string that is not a valid email address. The user agent should act in a manner consistent with expecting the user to provide a single email address. 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. 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 email address.

The value sanitization algorithm is as follows: Strip newlines from the value, then strip leading and trailing ASCII whitespace from the value.

Constraint validation: While the value of the element is neither the empty string nor a single valid email 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 email 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 email 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 email 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 ASCII 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 email address list.

The value sanitization algorithm is as follows:
1. **Split on commas** the element's `value`\(^{566}\), strip leading and trailing ASCII whitespace from each resulting token, if any, and let the element's `values`\(^{566}\) be the (possibly empty) resulting list of (possibly empty) tokens, maintaining the original order.

2. Let the element's `value`\(^{566}\) be the result of concatenating the element's `values`\(^{566}\), separating each value from the next by a single U+002C COMMA character (,), maintaining the list's order.

**Constraint validation:** While the `value`\(^{566}\) of the element is not a [valid email address list]\(^{563}\), the element is suffering from a type mismatch\(^{569}\).

When the `multiple`\(^{524}\) attribute is set or removed, the user agent must run the [value sanitization algorithm]\(^{467}\).

A **valid email 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]\(^{1270}\) [RFC5322]\(^{1276}\) [RFC1034]\(^{1276}\) [RFC1123]\(^{1276}\)

```plaintext
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 >

Note
This requirement is a willful violation\(^{527}\) of RFC 5322, which defines a syntax for email 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.

Note
The following JavaScript- and Perl-compatible regular expression is an implementation of the above definition.

```regex
/^[a-zA-Z0-9.\#$%&\*\+/=?\^\`\{\}\|\-\_]@[a-zA-Z0-9\-\.\_\{\}\]@[a-zA-Z0-9\-\.\_]\{0,61\}[a-zA-Z0-9]\{0,61\}[a-zA-Z0-9\-\.\_]\{0,61\}[a-zA-Z0-9\-]\}*$
```

A **valid email address list** is a set of comma-separated tokens\(^{487}\), where each token is itself a valid email address\(^{563}\). To obtain the list of tokens from a valid email address list\(^{567}\), an implementation must split the string on commas.

**Bookkeeping details**
- The following common `input`\(^{489}\) element content attributes, IDL attributes, and methods `apply`\(^{488}\) to the element: `autocomplete`\(^{577}\), `list`\(^{528}\), `maxlength`\(^{577}\), `minlength`\(^{576}\), `multiple`\(^{524}\), `pattern`\(^{576}\), `placeholder`\(^{576}\), `readonly`\(^{522}\), `required`\(^{523}\), and `size`\(^{522}\) content attributes; `list`\(^{534}\) and `value`\(^{550}\) IDL attributes; `select()`\(^{507}\) method.
- The `value`\(^{551}\) IDL attribute is in mode `value`\(^{552}\).
- The `input`\(^{512}\) and `change`\(^{512}\) events `apply`\(^{488}\).
- The following content attributes must not be specified and do not `apply`\(^{488}\) to the element: `accept`\(^{515}\), `alt`\(^{515}\), `checked`\(^{552}\), `dirname`\(^{550}\), `formaction`\(^{571}\), `formenctype`\(^{577}\), `formmethod`\(^{577}\), `formnovalidate`\(^{577}\), `formtarget`\(^{577}\), `height`\(^{550}\), `max`\(^{515}\), `min`\(^{515}\), `multiple`\(^{524}\), `maxlength`\(^{577}\), `minlength`\(^{576}\), `multiple`\(^{524}\), `pattern`\(^{576}\), `src`\(^{526}\), `step`\(^{522}\), and `width`\(^{514}\).
- The following IDL attributes and methods do not `apply`\(^{488}\) to the element: `checked`\(^{552}\), `files`\(^{523}\), `selectionStart`\(^{522}\), `selectionEnd`\(^{522}\), `selectionDirection`\(^{540}\), `valueAsNumber`\(^{540}\), and `valueAsNumber`\(^{540}\) IDL attributes; `setMaxRangeText()`\(^{548}\), `setSelectionRange()`\(^{548}\), `stepDown()`\(^{553}\), and `stepUp()`\(^{553}\) methods.

### 4.10.5.1.6 Password state (type=password) \(^{560}\)

When an `input`\(^{489}\) element's `type`\(^{545}\) attribute is in the `Password`\(^{563}\) state, the rules in this section apply.

The `input`\(^{489}\) element `represents`\(^{512}\) a one line plain text edit control for the element's `value`\(^{566}\). The user agent should obscure the value so that people other than the user cannot see it.

If the element is `mutable`\(^{566}\), its `value`\(^{566}\) 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`\(^{566}\).

The `value`\(^{467}\) 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 newlines from the value.

Bookkeeping details

- The following common input element content attributes, IDL attributes, and methods apply to the element: autocomplete, maxlength, minlength, pattern, placeholder, required, and size content attributes; selectionStart, selectionEnd, selectionDirection, and value IDL attributes; select(), setRangeText(), and setSelectionRange() methods.

- The value 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, dir, 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)  

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.

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.

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, as used in the other algorithms). 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.
The `Date` state (and other date- and time-related states described in subsequent sections) is not intended for the entry of values for which a precise date and time relative to the contemporary calendar cannot be established. For example, it would be inappropriate for the entry of times like "one millennium after the big bang", "the early part of the Jurassic period", or "a winter around 250 BCE".

For the input of dates before the introduction of the Gregorian calendar, authors are encouraged to not use the `Date` state (and other date- and time-related states described in subsequent sections). As user agents are not required to support converting dates and times from earlier periods to the Gregorian calendar, and asking users to do so manually puts an undue burden on users. (This is complicated by the manner in which the Gregorian calendar was phased in, which occurred at different times in different countries, ranging from partway through the 16th century all the way to early in the 20th.) Instead, authors are encouraged to provide fine-grained input controls using the `select` element and `input` elements with the `Number` state.

### Bookkeeping details
- The following common `input` element content attributes, IDL attributes, and methods apply to the element: `autocomplete`, `list`, `max`, `min`, `required`, `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`, `dir`, `dirname`, `formaction`, `formmethod`, `formnovalidate`, `formtarget`, `height`, `maxlength`, `minlength`, `multiple`, `pattern`, `placeholder`, `size`, `src`, `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)  

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. 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 the [introduction section](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#type месяц) for a discussion of the difference between the input format and submission format for date, time, and number form controls, and the [implementation notes](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#type месяц) 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 (there is no conversion needed as the algorithms use months). 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, formmethod, formnovalidate, formtarget, height, minheight, minlength, 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)

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.

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 the introduction section 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, as used in the other algorithms). The default step is 1 week. The default step base is −259,200,000 (the start of week 1970-W01).

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** is as follows: If parsing a week from input results in an error, then return an error; otherwise, return a new Date representing midnight UTC on the Monday of the parsed week.

The **algorithm to convert a Date object to a string** 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()* 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*, *formmethod*, *formnovalidate*, *formtarget*, *height*, *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)

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 the introduction section 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* 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, as used in the other algorithms). 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** 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** is as follows: Return a valid time string that represents the *time* after midnight on a day with no time changes.

The **algorithm to convert a string to a Date object** is as follows: If parsing a time from input results in an error, then return an error; otherwise, return a new Date representing the parsed time in UTC on 1970-01-01.
The algorithm to convert a Date object to a string⁴⁹⁷, given a Date object input, is as follows: Return a valid time string⁵⁰⁶ that represents the UTC time⁷⁶ component that is represented by input.

**Bookkeeping details**

- The following common input⁴⁹⁵ element content attributes, IDL attributes, and methods apply⁴⁹⁶ to the element: autocomplete⁵⁷⁷, list⁵⁰⁹, max⁵ⁱ⁴, min⁵⁵⁶, readonly⁵⁵⁲, required⁵⁵⁳, and step⁵⁵⁵, content attributes; list⁵¹⁹, value⁵³⁷, valueAsDate⁵³⁸, and valueAsNumber⁵³⁹ IDL attributes; select()⁵⁵⁵, stepDown()⁵⁵⁵, and stepUp()⁵⁵⁵, methods.

- The value⁵³², IDL attribute is in mode value⁵³².

- The input⁵¹⁹⁶ and changes⁵¹⁹⁶ events apply⁴⁹⁶.

- The following content attributes must not be specified and do not apply⁴⁹⁶ to the element: accept⁵¹⁹, alt⁵¹⁹, checked⁴⁹⁷, dirname⁵⁶⁰, formaction⁵⁷⁷, formcontrol⁵⁷⁷, formenctype⁵⁷⁷, formmethod⁵⁷⁷, formnovalidate⁵⁷⁷, formtarget⁵³⁷, height⁵⁶⁰, max⁵⁶⁰, min⁵⁶⁰, maxlength⁵⁵⁵, minlength⁵⁵⁵, multiple⁵⁵⁵, pattern⁵⁵⁵, placeholder⁵³⁰, size⁵³², src⁵³¹, and width⁵⁵⁵.

- The following IDL attributes and methods do not apply⁴⁹⁶ to the element: checked⁵³¹, files⁵³¹, selectionStart⁵³⁷, selectionEnd⁵³⁷, and selectionDirection⁵³⁸ IDL attributes; setRangeText()⁵³⁸, and setSelectionRange()⁵³⁸ methods.

4.10.5.11 Local Date and Time state (type=dateTime-local) §⁶⁵⁰

When an input⁴⁹³ element's type⁴⁹⁵ attribute is in the Local Date and Time⁵⁵⁸ state, the rules in this section apply.

The input⁴⁹³ element represents⁵²⁴ a control for setting the element's value⁵⁶⁶ to a string representing a local date and time⁷⁷, with no time-zone offset information.

If the element is mutable⁵⁶⁶, the user agent should allow the user to change the date and time⁷⁷ represented by its value⁵⁶⁶, as obtained by parsing a date and time⁷⁷ from it. User agents must not allow the user to set the value⁵⁶⁶ to a non-empty string that is not a valid normalized local date and time string⁷⁷. If the user agent provides a user interface for selecting a local date and time⁷⁷, then the value⁵⁶⁶ must be set to a valid normalized local date and time string⁷⁷ representing the user’s selection. User agents should allow the user to set the value⁵⁶⁶ to the empty string.

**Constraint validation:** While the user interface describes input that the user agent cannot convert to a valid normalized local date and time string⁷⁷, the control is suffering from bad input⁵⁰⁹.

**Note**

See the introduction section⁴⁸⁸ for a discussion of the difference between the input format and submission format for date, time, and number form controls, and the implementation notes⁵²⁴ regarding localization of form controls.

The value⁵⁶⁷ attribute, if specified and not empty, must have a value that is a valid local date and time string⁹⁷⁷.

The value sanitization algorithm⁴⁹⁷ is as follows: If the value⁵⁶⁶ of the element is a valid local date and time string⁷⁷, then set it to a valid normalized local date and time string⁹⁷⁷ representing the same date and time; otherwise, set it to the empty string instead.

The min⁵²⁶ attribute, if specified, must have a value that is a valid local date and time string⁷⁷. The max⁵²⁶ attribute, if specified, must have a value that is a valid local date and time string⁷⁷.

The step⁵²⁷ attribute is expressed in seconds. The step scale factor⁵²⁷ is 1000 (which converts the seconds to milli­seconds, as used in the other algorithms). The default step⁵²⁷ is 60 seconds.

When the element is suffering from a step mismatch⁵⁵⁹, the user agent may round the element’s value⁵⁶⁶ to the nearest local date and time⁷⁷ for which the element would not suffer from a step mismatch⁵⁵⁹.

The algorithm to convert a string to a number⁴⁹⁷, given a string input, is as follows: If parsing a date and time⁷⁸ 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 "1970-01-01T00:00:00.000Z") to the parsed local date and time⁷⁷, ignoring leap seconds.

The algorithm to convert a number to a string⁴⁹⁷, given a number input, is as follows: Return a valid normalized local date and time string⁷⁷ 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 "1970-01-01T00:00:00.000Z").

**Note**

See the note on historical dates⁵⁵⁵ in the Date⁵⁵⁴ state section.
4.10.5.1.12 Number state (type=number) 

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 instead.

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.

Note

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 Romans might display the value in Roman numerals rather than in decimal; or (more realistically) 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.

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\(^{p527}\) is 1. The default step\(^{p527}\) is 1 (allowing only integers to be selected by the user, unless the step base\(^{p527}\) has a non-integer value).

When the element is suffering from a step mismatch\(^{p591}\), the user agent may round the element’s value\(^{p566}\) to the nearest number for which the element would not suffer from a step mismatch\(^{p591}\). 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\(^{p497}\), given a string input, is as follows: If applying the rules for parsing floating-point number values\(^{p70}\) to input results in an error, then return an error; otherwise, return the resulting number.

The algorithm to convert a number to a string\(^{p497}\), given a number input, is as follows: Return a valid floating-point number\(^{p69}\) that represents the value.

**Bookkeeping details**

The following common input\(^{p497}\) element content attributes, IDL attributes, and methods apply\(^{p497}\) to the element: autocomplete\(^{p573}\), list\(^{p573}\), max\(^{p573}\), min\(^{p573}\), placeholder\(^{p528}\), readonly\(^{p528}\), required\(^{p528}\), and step\(^{p528}\) content attributes; list\(^{p574}\), value\(^{p574}\), and valueAsNumber\(^{p571}\) IDL attributes; select\(^{p571}\), stepDown\(^{p571}\), and stepUp\(^{p571}\) methods.

The value\(^{p571}\) IDL attribute is in mode value\(^{p573}\).

The input\(^{p497}\) and change\(^{p129}\) events apply\(^{p496}\).

The following content attributes must not be specified and do not apply\(^{p496}\) to the element: accept\(^{p572}\), alt\(^{p572}\), checked\(^{p572}\), dirname\(^{p528}\), formation\(^{p572}\), formenctype\(^{p572}\), formmethod\(^{p572}\), formnovalidate\(^{p572}\), formtarget\(^{p525}\), height\(^{p528}\), maxlength\(^{p528}\), min\(^{p528}\), multiple\(^{p528}\), pattern\(^{p528}\), size\(^{p528}\), src\(^{p528}\), and width\(^{p528}\).

The following IDL attributes and methods do not apply\(^{p496}\) to the element: checked\(^{p572}\), files\(^{p572}\), selectionStart\(^{p572}\), selectionEnd\(^{p572}\), selectionDirection\(^{p572}\), and valueAsDate\(^{p571}\) IDL attributes; setRangeText\(^{p591}\), and setSelectionRange\(^{p596}\) methods.

**Example**

Here is an example of using a numeric input control:

```html
<label>How much do you want to charge? $</label>
<input type="number" min="0" step="0.01" name="price"/>
```

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 an inputmode\(^{p576}\) or pattern\(^{p525}\) attribute).

### 4.10.5.1.13 Range state (type=range) \(^{p51}\)

When an input\(^{p493}\) element’s type\(^{p495}\) attribute is in the Range\(^{p510}\) state, the rules in this section apply.

The input\(^{p493}\) element represents\(^{p124}\) a control for setting the element’s value\(^{p566}\) to a string representing a number, but with the caveat that the exact value is not important, letting UAs provide a simpler interface than they do for the Number\(^{p529}\) state.

If the element is mutable\(^{p566}\), the user agent should allow the user to change the number represented by its value\(^{p566}\), as obtained from applying the rules for parsing floating-point number values\(^{p70}\) to it. User agents must not allow the user to set the value\(^{p544}\) to a string that is not a valid floating-point number\(^{p69}\). If the user agent provides a user interface for selecting a number, then the value\(^{p566}\) must be set to a best representation of the number representing the user’s selection as a floating-point number\(^{p70}\). User agents must not allow the user to set the value\(^{p566}\) to the empty string.

**Constraint validation:** While the user interface describes input that the user agent cannot convert to a valid floating-point number\(^{p69}\), the control is suffering from bad input\(^{p591}\).

The value\(^{p497}\) attribute, if specified, must have a value that is a valid floating-point number\(^{p69}\).
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 **default value**.

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.

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.

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:

```html
<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:

```css
input { height: 75px; width: 49px; background: #D5CCBB; color: black; }
```

...might render as:

Note how the UA 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 UA 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.

For another example, consider the following markup fragment:

```html
<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:
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`, `maxlength`, `minlength`, `multiple`, `pattern`, `placeholder`.

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**

Finally, here is an example of a range control with two labeled values:

```html
code<input type="range" name="a" list="a-values">
code<datalist id="a-values">
code<option value="10" label="Low">
code<option value="90" label="High">
</datalist>
```

With styles that make the control draw vertically, it might look as follows:

![Vertical range control](image)

---

**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`.

---
4.10.5.1.14 Color state (type=color) §p81

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 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; select() method.

The value IDL attribute is in mode.

The input and change events apply.

The following content attributes must not be specified and do not apply to the element: accept, alt, checked, dir, formaction, formenctype, formmethod, formnovalidate, formtarget, height, max, maxsize, 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; selectRangeText(), setSelectionRange(), and stepUp() methods.

4.10.5.1.15 Checkbox state (type=checkbox) §p81

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 checked state. If the element’s checked 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.

The input activation behavior is to run the following steps:

1. If the element is not connected, then return.
2. Fire an event named at the element with the bubbles and composed attributes initialized to true.
3. Fire an event named at the element with the bubbles attribute initialized to true.
Constraint validation: If the element is `required` and its `checkedness` is false, then the element is suffering from being missing.

**For web developers (non-normative)**

```plaintext
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`.
- 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`, `formnovalidate`, `formtarget`, `height`, `list`, `max`, `maxlength`, `min`, `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`, `select`, `setRangeText`, `setSelectionRange`, `stepDown`, and `stepUp` methods.

### 4.10.5.1.16 Radio Button state (type=radio)

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 checked state set to true. If the element's checked 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 tree.
- They both have a `name` attribute, their `name` attributes are not empty, and the value of a's `name` attribute equals the value of b's `name` attribute.

A tree 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 checked state is true after the occurrence, the checked state of all the other elements in the same radio button group must be set to false:

- The element's checked 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.

The `input activation behavior` is to run the following steps:

1. If the element is not connected, then return.
2. Fire an event named `input` at the element with the `bubbles` and `composed` attributes initialized to true.
3. Fire an event named `change` at the element with the `bubbles` attribute initialized to true.

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 checked state that is false, then the element is suffering from being missing.
Example

The following example, for some reason, has specified that puppets are both required\(^{523}\) and disabled\(^{570}\):

```
<form>
  <p><label><input type="radio" name="dog-type" value="pupper" required disabled> Pupper</label>
  <label><input type="radio" name="dog-type" value="doggo"> Doggo</label>
  <button>Make your choice</button>
</form>
```

If the user tries to submit this form without first selecting "Doggo", then both input\(^{593}\) elements will be suffering from being missing\(^{590}\), since an element in the radio button group\(^{531}\) is required\(^{523}\) (viz. the first element), and both of the elements in the radio button group have a false checkedness\(^{566}\).

On the other hand, if the user selects "Doggo" and then submits the form, then neither input\(^{593}\) element will be suffering from being missing\(^{590}\), since while one of them is required\(^{523}\), not all of them have a false checkedness\(^{566}\).

Note

If none of the radio buttons in a radio button group\(^{524}\) are checked, 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\(^{495}\) element content attributes and IDL attributes apply\(^{496}\) to the element: checked\(^{497}\) and required\(^{513}\) content attributes; checked\(^{533}\) and value\(^{533}\) IDL attributes.
- The value\(^{531}\) IDL attribute is in mode default\(^{532}\).
- The input\(^{512}\) and change\(^{526}\) events apply\(^{496}\).
- The following content attributes must not be specified and do not apply\(^{496}\) to the element: accept\(^{532}\), alt\(^{551}\), autocomplete\(^{577}\), dirname\(^{565}\), formaction\(^{571}\), formenctype\(^{531}\), formmethod\(^{521}\), formnovalidate\(^{572}\), size\(^{532}\), src\(^{519}\), step\(^{527}\), and width\(^{560}\).
- The following IDL attributes and methods do not apply\(^{496}\) to the element: files\(^{511}\), list\(^{526}\), selectionStart\(^{547}\), selectionEnd\(^{547}\), selectionDirection\(^{518}\), values\(^{535}\), and valueAsNumber\(^{513}\) IDL attributes; select\(^{512}\), setRangeText\(^{521}\), setSelection\(^{519}\), stepDown\(^{524}\), and stepUp\(^{515}\) methods.

---

**4.10.5.1.17 File Upload state (type=file)**

When an input\(^{495}\) element's type\(^{495}\) attribute is in the File Upload\(^{515}\) state, the rules in this section apply.

The input\(^{495}\) element represents\(^{514}\) 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\(^{515}\), 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\(^{515}\) state, are those parts of file names that are separated by U+005C REVERSE SOLIDUS character (\) characters.

Unless the multiple\(^{524}\) attribute is set, there must be no more than one file in the list of selected files\(^{515}\).

The element's input activation behavior\(^{498}\) is to run the following steps:

1. If the algorithm is invoked when the element's Window\(^{427}\) object does not have transient activation\(^{569}\), then return without doing anything else.
2. Run these steps in parallel\(^{441}\):
   1. Optionally, wait until any prior execution of this algorithm has terminated.
   2. Display a prompt to the user requesting that the user specify some files. If the multiple\(^{534}\) 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.
   3. Wait for the user to have made their selection.
3. Update the file selection\(^{516}\) for the input\(^{493}\) 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 update the file selection for the element.

If the element is not mutable, the user agent must not allow the user to change the element's selection.

To update the file selection for an element:

1. Queue an element task on the user interaction task source given element and the following steps:
   1. Update element's selected files so that it represents the user's selection.
   2. Fire an event named input at the input element, with the bubbles and composed attributes initialized to true.
   3. Fire an event named change at the input element, with the bubbles attribute initialized to true.

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 string 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**

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

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:

```javascript
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:

```html
<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`, `formation`, `formenctype`, `formmethod`, `formnovalidate`, `formtarget`, `height`, `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) §p512

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.

The element's input 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.

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, list, max, min, maxlength, 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(), stepUp(), and stepDown() methods.
- The input and change events do not apply.

4.10.5.1.19 Image Button state (type=image) §p512

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, then:

1. Let request be a new request whose url is the resulting URL record\(^{889}\), client is the element's node document's relevant settings object\(^{902}\), destination is "image", credentials mode is "include", and whose use-URL-credentials flag is set.

2. Fetch request.

Fetching the image must delay the load event\(^{8157}\) of the element's node document until the task\(^{991}\) that is queued\(^{992}\) by the networking task source\(^{992}\) 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\(^{991}\) that is queued\(^{992}\) by the networking task source\(^{992}\) 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\(^{990}\) 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\(^{990}\).

User agents must not support non-image resources with the input\(^{949}\) element. User agents must not run executable code embedded in the image resource. User agents must only display the first page of a multipart resource. User agents must not allow the resource to act in an interactive fashion, but should honor any animation in the resource.

The task\(^{991}\) that is queued\(^{992}\) by the networking task source\(^{992}\) once the resource has been fetched, must, if the download was successful and the image is available\(^{819}\), queue an element task\(^{992}\) on the user interaction task source\(^{992}\) given the input\(^{949}\) element to fire an event named load\(^{1208}\) at the input\(^{495}\) 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 an element task\(^{992}\) on the user interaction task source\(^{992}\) given the input\(^{493}\) element to fire an event named error\(^{1206}\) on the input\(^{949}\) element.

The alt attribute provides the textual label for the button for users and user agents who cannot use the image. The alt\(^{519}\) 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\(^{493}\) element supports dimension attributes\(^{945}\).

If the src\(^{493}\) attribute is set, and the image is available\(^{819}\) and the user agent is configured to display that image, then: The element represents\(^{812}\), a control for selecting a coordinate\(^{519}\) from the image specified by the src\(^{493}\) attribute; if the element is mutable\(^{556}\), the user agent should allow the user to select this coordinate\(^{519}\), and the element's input activation behavior\(^{448}\) is as follows: if the element has a form owner\(^{556}\), and the element's node document is fully active\(^{681}\), take the user's selected coordinate\(^{519}\) and submit\(^{556}\) the input\(^{493}\) element's form owner\(^{556}\) from the input\(^{493}\) element. If the user activates the control without explicitly selecting a coordinate, then the coordinate (0,0) must be assumed.

Otherwise, the element represents\(^{812}\), a submit button whose label is given by the value of the alt\(^{519}\) attribute; the element's input activation behavior\(^{448}\) is as follows: if the element has a form owner\(^{556}\), and the element's node document is fully active\(^{681}\), set the selected coordinate\(^{519}\) to (0,0), and submit\(^{556}\) the input\(^{497}\) element's form owner\(^{556}\) from the input\(^{493}\) element.

In either case, if the element has no form owner\(^{556}\) or the element's node document is not fully active\(^{681}\), then its input activation behavior\(^{448}\) must be to do nothing..

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\(^{446}\), 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\(^{446}\), 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 `formaction`, `formenctype`, `formmethod`, `formnovalidate`, and `formtarget` attributes are attributes for form submission.

**Bookkeeping details**

- The following common `input` element content attributes and IDL attributes apply to the element: `alt`, `formaction`, `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`, `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**

Take the following form:

```html
<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)

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.

The element's `input 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.
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, 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, valueAsNumber, and IDL attributes: select(), setRangeText(), setSelectionRange(), stepDown(), and stepUp().
- The and change events do not apply.

### 4.10.5.21 Button state (type=button)

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.

The element has no input 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, 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, valueAsNumber, and IDL attributes: select(), setRangeText(), setSelectionRange(), stepDown(), and stepUp().
- The and change events do not apply.

### 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**

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 later, 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 does not 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.

The `minlength` attribute, when it applies, is a form control `minlength` attribute.

If the `input` element has a `maximum allowed value length`, then the `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**

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.

```html
<label>What are you doing? <input name=status maxlength=140></label>
```

**Example**

Here, a password is given a minimum length:

```html
<p><label>Username: <input name=u required></label></p>
<p><label>Password: <input name=p required minlength=12></label></p>
```

### 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 can still function, whereas disabled controls generally do not function as controls until they are enabled. This is spelled out in more detail elsewhere in this specification with normative requirements that refer to the `disabled` concept (for example, the element's activation behavior, whether or not it is a focusable area, or when constructing the entry list). Any other behavior related to user interaction with disabled controls, such as whether text can be selected or copied, is not defined in this standard.

Only text controls can be made read-only, since for other controls (such as checkboxes and buttons) there is no useful distinction between being read-only and being disabled, so the `readonly` attribute does not apply.

**Example**

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).
4.10.5.3.4 The **required** attribute

The **required** attribute is a [boolean attribute](https://www.w3.org/TR/html5/forms.html#boolean-attribute). When specified, the element is **required**.

**Constraint validation:** If the element is **required**, and its **value** IDL attribute [applies](https://www.w3.org/TR/html5/forms.html#apply) and is in the mode **value**, and the element is [mutable](https://www.w3.org/TR/html5/forms.html#mutable), and the element’s **value** is the empty string, then the element is suffering from being missing.

**Example**

The following form has two required fields, one for an email 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.

```html
<html>
<body>
  <form method="post" action="/newaccount" id="newaccount">
    <p><label for="username">Email address:</label><br>
       <input type="email" required name="un" id="username" value=""></p>
    <p><label for="password1">Password:</label><br>
       <input type="password" required name="up" id="password1" value=""></p>
    <p><label for="password2">Confirm password:</label><br>
       <input type="password" name="up2" id="password2" value=""></p>
    <p><input type="submit" value="Create account"></p>
  </form>
</body>
</html>
```

**Example**

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:

```html
<html>
<body>
  <form method="post" action="/products.cgi" enctype="multipart/form-data">
    <table>
      <tr><th>Product ID</th><th>Product name</th><th>Price</th><th>Action</th></tr>
      <tr><td><input type="text" value="H412"></td><td><input type="text" value="Table lamp Ulke"></td><td>$49.99</td><td>
         <button formnovalidate="formnovalidate" name="action" value="delete:1">Delete</button></td></tr>
      <tr><td><input type="text" value="FG28"></td><td><input type="text" value="Table lamp Ulke"></td><td>$24.99</td><td>
         <button formnovalidate="formnovalidate" name="action" value="delete:2">Delete</button></td></tr>
      <tr><td><input type="text" value=""></td><td><input type="text" value=""></td><td></td><td>
         <button formnovalidate="formnovalidate" name="action" value="delete:3">Delete</button></td></tr>
    </table>
  </form>
</body>
</html>
```
The `multiple` attribute is a boolean attribute that indicates whether the user is to be allowed to specify more than one value.

```
<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>
  <p><label><input type="radio" name="bechdel" value="no-names"> No, the female characters never talk to each other. </label></p>
  <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>
  <p><label><input type="radio" name="bechdel" value="yes" required> Yes. </label></p>
  <p><label><input type="radio" name="bechdel" value="unknown"> I don't know. </label></p>
</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**

The following extract shows how an email client's "To" field could accept multiple email addresses.

```
<label>To: <input type=email multiple name=to></label>
```

If the user had, amongst many friends in their user contacts database, two friends "Spider-Man" (with address "spider@parker.example.net") and "Scarlet Witch" (with address "scarlet@avengers.example.net"), then, after the user has typed "s", the user agent might suggest these two email addresses to the user.

```
To: s ▼

spider@parker.example.net Spider-Man
scarlet@avengers.example.net Scarlet Witch
```

The page could also link in the user's contacts database from the site:

```
<label>To: <input type=email multiple name=to 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 control, and then started typing a second email address starting with "s". The user agent might show both the two friends mentioned earlier, as well as the "astrophy" and "astronomy" values given in the `datalist` element.
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 [+U, +N]` production.

The compiled pattern regular expression of an `input` element, if it exists, is a JavaScript `RegExp` object. It is determined as follows:

1. If the element does not have a pattern attribute specified, then return nothing. The element has no compiled pattern regular expression.
2. Let `pattern` be the value of the pattern attribute of the element.
3. Let `regexpCompletion` be `RegExpCreate(pattern, "u")`.
4. If `regexpCompletion` is an abrupt completion, then return nothing. The element has no compiled pattern regular expression.
5. Let `anchoredPattern` be the string `"^(?::", followed by `pattern`, followed by ")$"`.
6. Return `RegExpCreate(anchoredPattern, "u")`.

A `RegExp` object `regexp` matches a string input, if `! RegExpBuiltinExec(regexp, input)` is not null.

**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's attribute's current state, and the element has a compiled pattern regular expression but that regular expression does not match 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 each of the element's `values`, then the element is suffering from a pattern mismatch.
When an input element has a pattern attribute specified, authors should 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.

Example
For example, the following snippet:

```html
<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 UA 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
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.

UAs 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 title as if an error has necessarily occurred.

4.10.5.3 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
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.

Constraint validation: When an element has a reversed range 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 and less than the minimum, the element is suffering from an underflow and suffering from an overflow.

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 attribute does not apply, then there is no allowed value step.
2. Otherwise, if the attribute is absent, then the allowed value step is the default step multiplied by the step scale factor.
3. Otherwise, if the attribute's value is an ASCII case-insensitive match for the string "any", then there is no allowed value step.
4. Otherwise, if the rules for parsing floating-point number values, when they are applied to the attribute's value, return an error, zero, or a number less than zero, then the allowed value step is the default step multiplied by the step scale factor.
5. Otherwise, the allowed value step is the number returned by the rules for parsing floating-point number values when they are applied to the attribute's value, multiplied by the step scale factor.

The step base is the value returned by the following algorithm:
1. If the element has a \texttt{min} content attribute, and the result of applying the \textit{algorithm to convert a string to a number} to the value of the \texttt{min} content attribute is not an error, then return that result.

2. If the element has a \texttt{value} content attribute, and the result of applying the \textit{algorithm to convert a string to a number} to the value of the \texttt{value} content attribute is not an error, then return that result.

3. If a \texttt{default step base} is defined for this element given its \texttt{type} attribute's state, then return it.

4. Return zero.

\textbf{Constraint validation}: When the element has an \texttt{allowed value step}, and the result of applying the \textit{algorithm to convert a string to a number} to the string given by the element's \texttt{value} is a number, and that number subtracted from the \texttt{step base} is not an integral multiple of the \texttt{allowed value step}, the element is \textit{suffering from a step mismatch}.

\textbf{Example}

The following range control only accepts values in the range 0..1, and allows 256 steps in that range:

\begin{verbatim}
<input name=\textit{opacity} type=range min=0 max=1 step=0.00392156863>
\end{verbatim}

\textbf{Example}

The following control allows any time in the day to be selected, with any accuracy (e.g. thousandth-of-a-second accuracy or more):

\begin{verbatim}
<input name=\textit{favtime} type=time step=any>
\end{verbatim}

Normally, time controls are limited to an accuracy of one minute.

\textbf{4.10.5.3.9 The list attribute} \footnote{8}

The \texttt{list} attribute is used to identify an element that lists predefined options suggested to the user.

If present, its value must be the \texttt{ID} of a \texttt{datalist} element in the same \texttt{tree}.

The \texttt{suggestions source element} is the first element in the \texttt{tree} in \texttt{tree order} to have an \texttt{ID} equal to the value of the \texttt{list} attribute, if that element is a \texttt{datalist} element. If there is no \texttt{list} attribute, or if there is no element with that \texttt{ID}, or if the first element with that \texttt{ID} is not a \texttt{datalist} element, then there is no \texttt{suggestions source element}.

If there is a \texttt{suggestions source element}, then, when the user agent is allowing the user to edit the \texttt{input} element's \texttt{value}, the user agent should offer the suggestions represented by the \texttt{suggestions source element} to the user in a manner suitable for the type of control used. If appropriate, the user agent should use the suggestion's \texttt{label} and \texttt{value} to identify the suggestion to the user.

User agents are encouraged to filter the suggestions represented by the \texttt{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. If filtering based on the user's input, user agents should use substring matching against both the suggestions' \texttt{label} and \texttt{value}.

\textbf{Example}

This text field allows you to choose a type of JavaScript function.

\begin{verbatim}
<input type=\textit{text} list=\textit{function-types}>
<datalist id=\textit{function-types}>
  <option value=\textit{function} >function</option>
  <option value=\textit{async function} >async function</option>
  <option value=\textit{generator function} >generator function</option>
  <option value=\textit{arrow function} >arrow function</option>
  <option value=\textit{async =>} >async arrow function</option>
  <option value=\textit{async function*} >async generator function</option>
</datalist>
\end{verbatim}
For user agents that follow the above suggestions, both the `<label>` and `<value>` would be shown:

```html
function
async function
function*
=>
async =>
async function*
```

Then, typing "arrow" or "=>" would filter the list to the entries with labels "arrow function" and "async arrow function". Typing "generator" or "*" would filter the list to the entries with labels "generator function" and "async generator function".

**Note**

As always, user agents are free to make user interface decisions which are appropriate for their particular requirements and for the user's particular circumstances. However, this has historically been an area of confusion for implementers, web developers, and users alike, so we've given some "should" suggestions above.

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 themself.

- **If the element's `type` attribute is in the Email 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 themself 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 an implementation-defined manner.

If the `list` attribute does not apply, there is no `<suggestions source element>`.

**Example**

This URL field offers some suggestions.

```html
<p>Homepage: <input name=hp type=url list=hpurls></p>
</html>
```

Other URLs from the user's history might show also; this is up to the user agent.

**Example**

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.
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 an alternative to a label. For a longer hint or other advisory text, the title attribute is more appropriate.

User agents should present this hint to the user, after having stripped newlines 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.

```html
<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 UAs, then fallback content can be placed inside the datalist element, as follows:

```html
<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 UAs that don't support datalist. The options, on the other hand, will be detected by all UAs, 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 UAs (because it affects the select), but it will not have any effect on the input element in UAs 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 an alternative to a label. For a longer hint or other advisory text, the title attribute is more appropriate.

Note

These mechanisms are very similar but subtly different: the hint given by the control's label is shown at all times; the short hint given in the placeholder attribute is shown before the user enters a value; and the hint in the title attribute is shown when the user requests further help.

User agents should present this hint to the user, after having stripped newlines 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

Here is an example of a mail configuration user interface that uses the `placeholder` attribute:

```html
<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

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:

```html
<input name="t1" type="tel" placeholder="&#x202E;1الهاتفرقم">
<input name="t2" type="tel" placeholder="&#x202E;2الهاتفرقم">
```

For slightly more clarity, here's the same example using numeric character references instead of inline Arabic:

```html
<input name="t1" type="tel" placeholder="1رقم الهاتف">
<input name="t2" type="tel" placeholder="2رقم الهاتف">
```

### 4.10.5.4 Common `input` element APIs

For web developers (non-normative)

```javascript
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.
```

```javascript
input .checked [ = value ]
Returns the current `checkedness` of the form control.
Can be set, to change the `checkedness`.
```

```javascript
input .files [ = files ]
Returns a `FileList` object listing the `selected files` of the form control.
Returns null if the control isn't a file control.
Can be set to a `FileList` object to change the `selected files` of the form control. For instance, as the result of a drag-and-drop operation.
```

```javascript
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.
```

```javascript
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.
```
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, return the current value of the element.

On setting:
1. Let oldValue be the element's value.
2. Set the element's value to the new value.
3. Set the element's dirty value flag to true.
4. Invoke the value sanitization algorithm, if the element's type attribute's current state defines one.
5. If the element's value (after applying the value sanitization algorithm) is different from oldValue, and the element has a text entry cursor position, move the text entry cursor position to the end of the text control, unselecting any selected text and resetting the selection direction to "none".

default
On getting, if the element has a value content attribute, return that attribute's value; otherwise, return the empty string.

On setting, set the value of the element's value content attribute to the new value.

default/on
On getting, if the element has a value content attribute, return that attribute's value; otherwise, return the string "on".

On setting, set the value of the element's value content attribute to the new value.

filename
On getting, 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, empty the list of selected files; otherwise, 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]

On setting, it must run these steps:

1. If the IDL attribute does not apply or the given value is null, then return.
2. Replace the element's selected files with the given value.

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` DOMException; 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, 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` DOMException. Otherwise, if the new value is a Not-A-Number (NaN) value, then set the value of the element to the empty 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.
2. If the element has no allowed value step, then throw an `InvalidStateError` DOMException.
3. If the element has a minimum and a maximum, and the minimum is greater than the maximum, then return.
4. If the element has a minimum and a maximum, and there is no value greater than or equal to the element's minimum and less than or equal to the element's maximum that, when subtracted from the step base, is an integral multiple of the allowed value step, then return.
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 subtracted from the step base is not an integral multiple of the allowed value step, then set value to the nearest value that, when subtracted from the step base, is an integral multiple of the allowed value step, and that is less than value if the method invoked was the `stepDown()` method, and more than value otherwise.

Otherwise (value subtracted from the step base is an integral multiple of the allowed value step):

1. Let n be the argument.
2. Let delta be the allowed value step multiplied by n.
3. If the method invoked was the `stepDown()` method, negate delta.

4. Let value be the result of adding delta to value.

8. If the element has a minimum and value is less than that minimum, then set value to the smallest value that, when subtracted from the step base, is an integral multiple of the allowed value step, and that is more than or equal to minimum.

9. If the element has a maximum and value is greater than that maximum, then set value to the largest value that, when subtracted from the step base, is an integral multiple of the allowed value step, and that is less than or equal to maximum.

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 return.

```
Example

This ensures that invoking the `stepUp()` method on the input element in the following example does not change the value of that element:

```html
<input type=number value=1 max=0>
```

11. Let value be 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.

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 input 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 input 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 an element task on the user interaction task source, given the input element to fire an event named input at the input element, with the bubbles and composed attributes initialized to true, and any time the user commits the change, the user agent must queue an element task on the user interaction task source, given the input element to fire an event named change at the input element, with the bubbles attribute initialized to true.

```
Example

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 input 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 an element task on the user interaction task source, given the input element to first fire an event named input at the input element, with the bubbles and composed attributes initialized to true, and then fire an event named change at the input element, with the bubbles attribute initialized to true.
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.

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 might 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.

For input elements without a defined input 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 an element task on the user interaction task source given the input element to fire an event named input at the input element, with the bubbles and composed attributes initialized to true. The corresponding change event, if any, will be fired when the control loses focus.

Examples of a user changing the element's value would include the user typing into a text control, pasting a new value into the control, or undoing an edit in that control. Some user interactions do not cause changes to the value, e.g., hitting the “delete” key in an empty text control, or replacing some text in the control with text from the clipboard that happens to be exactly the same text.

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 an element task on the user interaction task source given the input element to first update the value accordingly, then fire an event named input at the input element, with the bubbles and composed attributes initialized to true, then fire an event named change at the input element, with the bubbles attribute initialized to true.

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.)

4.10.6 The button element

Categories:
- Flow content
- Phrasing content
- Interactive content
- Listed
- Labelable
- Submittable
- Autocapitalize-inheriting
- 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 and no descendant with the tabindex attribute specified.

Tag omission in text/html:
Neither tag is omissible.
Content attributes:

- **disabled** — Whether the form control is disabled
- **form** — Associates the element with a form element
- **formaction** — URL to use for form submission
- **formenctype** — Entry list encoding type to use for form submission
- **formmethod** — Variant to use for form submission
- **formnovalidate** — Bypass form control validation for form submission
- **formtarget** — Browsing context for form submission
- **name** — Name of the element to use for form submission and in the `form.elements` API
- **type** — Type of button
- **value** — Value to be used for form submission

Accessibility considerations:

- For authors.
- For implementers.

DOM interface:

```idl
[Exposed=Window]
interface HTMLButtonElement : HTMLElement {
    [LegacyConstructor] constructor();

    [CEReactions] attribute boolean disabled;
    readonly attribute HTMLFormElement? form;
    [CEReactions] attribute USVString formAction;
    [CEReactions] attribute DOMString formEnctype;
    [CEReactions] attribute DOMString formMethod;
    [CEReactions] attribute boolean formNoValidate;
    [CEReactions] attribute DOMString formTarget;
    [CEReactions] attribute DOMString name;
    [CEReactions] attribute DOMString type;
    [CEReactions] attribute DOMString value;

    readonly attribute boolean willValidate;
    readonly attribute ValidityState validity;
    readonly attribute DOMString validationMessage;
    boolean checkValidity();
    boolean reportValidity();
    undefined setCustomValidity(DOMString error);

    readonly attribute ModelList 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.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>State</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>submit</td>
<td>Submit Button</td>
<td>Submits the form.</td>
</tr>
<tr>
<td>reset</td>
<td>Reset Button</td>
<td>Resets the form.</td>
</tr>
<tr>
<td>button</td>
<td>Button</td>
<td>Does nothing.</td>
</tr>
</tbody>
</table>

The `missing value default` and `invalid value default` are 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 or the `Button` state, the element is barred from...
A **button** element's activation behavior is to run the steps defined in the following list for the current state of this element's **type** attribute, if this element is not **disabled**, and do nothing otherwise:

### 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.

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 **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.

The **value** IDL attribute must **reflect** the content attribute 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 **disabled**, **form**, and **name** IDL attributes are part of the element's forms API.

**Example**

The following button is labeled "Show hint" and pops up a dialog box when activated:

```html
<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
- Autocapitalize-inheriting
- Form-associated element

537
Palpable content\textsuperscript{134}.

Contexts in which this element can be used\textsuperscript{129}:
Where phrasing content\textsuperscript{133} is expected.

Content model\textsuperscript{129}:
Zero or more option\textsuperscript{545}, optgroup\textsuperscript{544}, and script-supporting\textsuperscript{134} elements.

Tag omission in text/html\textsuperscript{130}:
Neither tag is omissible.

Content attributes\textsuperscript{130}:
Global attributes\textsuperscript{137}
autocomplete\textsuperscript{523} — Hint for form autofill feature
disabled\textsuperscript{579} — Whether the form control is disabled
form\textsuperscript{548} — Associates the element with a form\textsuperscript{548} element
multiple\textsuperscript{539} — Whether to allow multiple values
name\textsuperscript{568} — Name of the element to use for form submission\textsuperscript{595} and in the form.elements\textsuperscript{595} API
required\textsuperscript{539} — Whether the control is required for form submission\textsuperscript{595}
size\textsuperscript{539} — Size of the control

Accessibility considerations\textsuperscript{130}:
If the element has a multiple\textsuperscript{539} attribute or a size\textsuperscript{539} attribute with a value \( \geq 1 \): for authors; for implementers.
Otherwise: for authors; for implementers.

DOM interface\textsuperscript{130}:

\begin{verbatim}
IDL
[Exposed=Window]
interface HTMLSelectElement : HTMLElement {
    [HTMLConstructor] constructor();

    [CEReactions] attribute DOMString autocomplete;
    [CEReactions] attribute boolean disabled;
    readonly attribute HTMLFormElement? form;
    [CEReactions] attribute boolean multiple;
    [CEReactions] attribute DOMString name;
    [CEReactions] attribute boolean required;
    [CEReactions] attribute unsigned long size;

    readonly attribute DOMString type;

    [SameObject] readonly attribute HTMLCollection options;
    [CEReactions] attribute unsigned long length;
    getter Element? item(unsigned long index);
    HTMLOptionElement? namedItem(DOMString name);
    [CEReactions] undefined add((HTMLOptionElement or HTMLOptGroupElement) element, optional (HTMLElement or long)? before = null);
    [CEReactions] undefined remove(); // ChildNode overload
    [CEReactions] undefined remove(long index);
    [CEReactions] setter undefined (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();
    undefined setCustomValidity(DOMString error);

    readonly attribute ModelList labels;

}\end{verbatim}
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 the 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 theoptgroup 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 `size` 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 dirtiness 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 `true`, and whenever an `option` element with its selectedness `true` 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 dirtiness 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 false.
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 `dirtiness` 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 an element task on the user interaction task source given the `select` element to run these steps:

1. Fire an event named `input` at the `select` element, with the `bubbles` and `composed` attributes initialized to true.
2. Fire an event named `change` at the `select` element, with the `bubbles` attribute initialized to true.

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 `dirtiness` 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 `autocomplete` attribute controls how the user agent provides autofill behavior.

A `select` element that is not `disabled` is `mutable`.

For web developers (non-normative)

`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` 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" `DOMException` 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**\[ = \text{value} \]

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**\[ = \text{value} \]

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 or set the value of an existing indexed property for a **select** element, it must instead run the corresponding algorithm on the **select** element's **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 is not 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 `disabled`, `form`, and `name` IDL attributes are part of the element's forms API.

**Example**

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.

```html
<p>
  <label for="unittype">Select unit type:</label>
  <select id="unittype" name="unittype">
    <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 placeholder can be used instead:

```html
<select name="unittype" 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**

Here, the user is offered a set of options from which they can select any number. By default, all five options are selected.

```html
<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**

Sometimes, a user has to select one or more items. This example shows such an interface.

```html
<label>
  Select the songs from that you would like on your Act II Mix Tape:
  <select multiple required name="act2">
    <option value="s1">It Sucks to Be Me (Reprise)
```
<option value="s2">There is Life Outside Your Apartment</option>
<option value="s3">The More You Ruv Someone</option>
<option value="s4">Schadenfreude</option>
<option value="s5">I Wish I Could Go Back to College</option>
<option value="s6">The Money Song</option>
<option value="s7">School for Monsters</option>
<option value="s8">The Money Song (Reprise)</option>
<option value="s9">There's a Fine, Fine Line (Reprise)</option>
<option value="s10">What Do You Do With a B.A. in English? (Reprise)</option>
<option value="s11">For Now</option>
</select>
</label>

### 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

**Accessibility considerations**: For authors, For implementers.

**DOM interface**: Unexposed interface `HTMLDataListElement`:

```javascript
[HTMLConstructor] constructor();

[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**

```html
<label>
  Animal:
  <input name=animal list=animals>
  <datalist id=animals>
    <option value="Cat">Cat</option>
    <option value="Dog">Dog</option>
  </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

```html
<label>
  Animal:
  <input name=animal list=animals>
</label>
<datalist id=animals>
  <label>
    or select from the list:
    <select name=animal>
      <option value="">
      <option> Cat
      <option> Dog
    </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`.

**For web developers (non-normative)**

```html
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 can 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

**Accessibility considerations:**

For authors.
For implementers.

DOM interface:

```idl
[Exposed=Window]
interface HTMLOptGroupElement : HTMLElement {
    [HTMLConstructor] constructor();

    [CEReactions] attribute boolean disabled;
    [CEReactions] 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` attribute is a boolean attribute and can be used to `disable` a group of `option` elements together.

The `label` 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` 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**

The following snippet shows how a set of lessons from three courses could be offered in a `select` drop-down widget:

```html
<form action="courseselector.dll" method="get">
<p>Which course would you like to watch today?<br />
<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 Magnestism">
<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 and is not a child of a `datalist` element: `Text` that is not `inter-element whitespace`.
If the element has no `label` attribute and is a child of a `datalist` element: `Text`.

Tag omission in text/html:
An `option` element's end tag can 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:
- `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 `form submission`

Accessibility considerations:
For authors.
For implementers.

DOM interface:
```javascript
[Exposed=Window,
 LegacyFactoryFunction=Option(optional DOMString text = "", optional DOMString value, optional boolean defaultSelected = false, optional boolean selected = false)]
interface HTMLOptionElement : HTMLElement {
    [HTMLConstructor] constructor();

    [CEReactions] attribute boolean disabled;
    readonly attribute HTMLFormElement? form;
    [CEReactions] attribute DOMString label;
    [CEReactions] attribute boolean defaultSelected;
    attribute boolean selected;
    [CEReactions] attribute DOMString value;

    [CEReactions] 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` 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` attribute provides a label for 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.

The `label` content attribute, if specified, must not be empty.

The `value` 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.

The selected attribute is a boolean attribute. It represents the default selectedness of the element.

The dirtiness 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 dirtiness is false, its selectedness must be set to true. Whenever an option element's selected attribute is removed, if its dirtiness 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.

For web developers (non-normative)

```javascript
function Option() {
  this.textContent = arguments[0] ? arguments[0] : '';  // TextContent constructor.
  this.text = this.textContent;  // Text constructor.
  this.form = null;  // Form constructor.
  this.selected = false;  // Boolean constructor.
  this.label = null;  // Label constructor.
  this.value = null;  // Value constructor.
  this.index = 0;  // Enum constructor.
}
```

A 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 dirtiness 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 ASCII 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 or SVG script elements.

The text attribute's setter must string replace all with the given value within this element.

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 legacy factory function is provided for creating HTMLOptionElement objects (in addition to the factory methods from DOM such as createElement()): Option(text, value, defaultSelected, selected). When invoked, the legacy factory function must perform the following steps:

1. Let document be the current global object's associated Document.
2. Let option be the result of creating an element given document, option, and the HTML namespace.
3. If text is not the empty string, then append to option a new Text node whose data is text.
4. If value is given, then set an attribute value for option using "value" and value.
5. If defaultSelected is true, then set an attribute value for option using "selected" and the empty string.
6. If selected is true, then set option's selectedness to true; otherwise set its selectedness to false (even if defaultSelected is true).
7. Return option.

4.10.11 The textarea element

Categories:
- Flow content
- Phrasing content
- Interactive content
- Listed, labelable, submittable, resettable, and autocapitalize-inheriting 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
- cols — Maximum number of characters per line
- dirname — Name of form control to use for sending the element's directionality in form submission
- disabled — Whether the form control is disabled
- form — Associates the element with a form element
- maxlength — Maximum length of value
- minlength — Minimum length of value
- name — Name of the element to use for 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 form submission
- rows — Number of lines to show
- wrap — How the value of the form control is to be wrapped for form submission
Accessibility considerations:
For authors.
For implementers.

DOM interface:

```
[Exposed=Window]
interface HTMLTextAreaElement : HTMLElement {
    [HTMLConstructor] constructor();

    [CEReactions] attribute DOMString autocomplete;
    [CEReactions] attribute unsigned long cols;
    [CEReactions] attribute DOMString dirName;
    [CEReactions] attribute boolean disabled;
    readonly attribute HTMLFormElement? form;
    [CEReactions] attribute long maxLength;
    [CEReactions] attribute long minLength;
    [CEReactions] attribute DOMString name;
    [CEReactions] attribute DOMString placeholder;
    [CEReactions] attribute boolean readOnly;
    [CEReactions] attribute boolean required;
    [CEReactions] attribute unsigned long rows;
    [CEReactions] attribute DOMString wrap;

    readonly attribute DOMString type;
    [CEReactions] attribute DOMString defaultValue;
    attribute [LegacyNullToEmptyString] DOMString value;
    readonly attribute unsigned long textLength;

    readonly attribute boolean willValidate;
    readonly attribute ValidityState validity;
    readonly attribute DOMString validationMessage;
    boolean checkValidity();
    boolean reportValidity();
    undefined setCustomValidity(DOMString error);

    readonly attribute NodeList labels;

    undefined select();
    attribute unsigned long selectionStart;
    attribute unsigned long selectionEnd;
    attribute DOMString selectionDirection;
    undefined setRangeText(DOMString replacement);
    undefined setRangeText(DOMString replacement, unsigned long start, unsigned long end, optional SelectionMode selectionMode = "preserve");
    undefined 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
In this example, a text control is marked read-only because it represents a read-only file:

```xml
<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" ] && exit;

...</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 an element task on the user interaction task source given the `textarea` element to fire an event named `input` at the `textarea` element, with the bubbles and composed attributes initialized to true. 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's `dirty value flag` must be set to true whenever the user interacts with the control in a way that changes the `raw value`.

The cloning steps for `textarea` elements must propagate the `raw value` and `dirty value flag` from the node being cloned to the copy.

The children changed steps for `textarea` elements must, if the element's `dirty value flag` is false, set the element's `raw value` to its child text content.

The reset algorithm for `textarea` elements is to set the `dirty value flag` back to false, and set the `raw value` of element to its child text content.

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 an element task on the user interaction task source given the `textarea` element to fire an event named `input` at the `textarea` element, with the bubbles and composed attributes initialized to true.

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\(^{550}\).

The `wrap` attribute is an enumerated attribute\(^{668}\) with two keywords and states: the `soft` keyword which maps to the `Soft\(^{551}\)` state, and the `hard` keyword which maps to the `Hard\(^{551}\)` state. The `missing value default\(^{668}\)` and `invalid value default\(^{668}\)` are the `Soft\(^{551}\)` state.

The `Soft` state indicates that the text in the `textarea\(^{548}\)` 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\(^{548}\)` is to have newlines added by the user agent so that the text is wrapped when it is submitted.

If the element's `wrap\(^{551}\)` attribute is in the `Hard\(^{551}\)` state, the `cols\(^{550}\)` attribute must be specified.

For historical reasons, the element's value is normalized in three different ways for three different purposes. The `raw value\(^{549}\)` is the value as it was originally set. It is not normalized. The `API value\(^{556}\)` is the value used in the `value\(^{552}\)` IDL attribute, `textarea wrapping transformation\(^{551}\)` applied to a string:

- 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.
- If the element's `wrap\(^{551}\)` attribute is in the `Hard\(^{551}\)` state, insert `U+000D CARRIAGE RETURN U+000A LINE FEED (CRLF)` character pairs into the string using an implementation-defined algorithm so that each line has no more than `character width\(^{550}\)` 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\(^{569}\)`.

If the `textarea\(^{548}\)` element has a `maximum allowed value length\(^{569}\)`, then the element's children must be such that the `length` of the value of the element's `descendant text content` with `newlines normalized` is equal to or less than the element's `maximum allowed value length\(^{569}\)`.

The `minlength` attribute is a form control `minlength attribute\(^{569}\)`.

The `required` attribute is a boolean attribute\(^{667}\). When specified, the user will be required to enter a value before submitting the form.

**Constraint validation:** If the element has its `required\(^{555}\)` attribute specified, and the element is `mutable\(^{566}\)`, and the element's `value\(^{566}\)` is the empty string, then the element is `suffering from being missing\(^{595}\)`.

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\(^{551}\)` attribute should not be used as an alternative to a `label\(^{498}\)`. For a longer hint or other advisory text, the `title\(^{516}\)` attribute is more appropriate.

Note

These mechanisms are very similar but subtly different: the hint given by the control's `label\(^{498}\)` is shown at all times; the short hint given in the `placeholder\(^{551}\)` attribute is shown before the user enters a value; and the hint in the `title\(^{516}\)` attribute is shown when the user requests further help.

User agents should present this hint to the user when the element's `value\(^{566}\)` is the empty string and the control is not `focused\(^{973}\)` (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.

If a user agent normally doesn't show this hint to the user when the control is `focused\(^{973}\)`, 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.

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 `autocomplete` attribute controls how the user agent provides autofill behavior.

For web developers (non-normative)

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` IDL 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 with fallback.

The `cols` IDL attribute's default value is 20. The `rows` IDL attribute's default value is 2. The `dirName` IDL attribute must reflect the `dirName` content attribute. 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` attribute's getter must return the element's `child text content`.

The `defaultValue` attribute's setter must `string replace all` with the given value within this element.

The `value` IDL attribute must, on getting, return the element's `API value`. On setting, it must perform the following steps:

1. Let `oldAPIValue` be this element's `API value`.
2. Set this element's `raw value` to the new value.
3. Set this element's `dirty value flag` to true.
4. If the new `API value` is different from `oldAPIValue`, then move the `text entry cursor position` to the end of the text control, unselecting any selected text and `resetting the selection direction` to "none".

The `textLength` IDL attribute must return the `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`, `setText()`, and `setSelectionRange()` methods and IDL attributes expose the element's text selection. The `disabled`, `form`, and `name` IDL attributes are part of the element's forms API.

Example

Here is an example of a `textarea` being used for unrestricted free-form text input in a form:

```html
<p>If you have any comments, please let us know: &lt;textarea cols=80 name=comments&gt;&lt;/textarea&gt;&lt;/p&gt;

To specify a maximum length for the comments, one can use the `maxLength` attribute:

```html
<p>If you have any short comments, please let us know: &lt;textarea cols=80 name=comments maxLength=200&gt;&lt;/textarea&gt;&lt;/p&gt;

To give a default value, text can be included inside the element:

```html
<p>If you have any comments, please let us know: &lt;textarea cols=80 name=comments&gt;You
```
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:

```html
<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:

```html
<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:

```html
<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 `output` element

**Categories**
- Flow content
- Phrasing content
- Listed
- labelable
- resettable
- autocapitalize-inheriting
- 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 element with a `form` element
  - `name` — Name of the element to use in the `form.elements` API.

**Accessibility considerations**
- For authors.
- For implementers.
The output element represents the result of a calculation performed by the application, or the result of a user action.

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, none of which are identical to another token and each of which must have the value of an ID of an element in the same tree.

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 default value override (null or a string). Initially it must be null.

The element's default value is determined by the following steps:

1. If this element's default value override is non-null, then return it.
2. Return this element's descendant text content.

The reset algorithm for output elements is to run these steps:

1. String replace all with this element's default value within this element.
2. Set this element's default value to null.

For web developers (non-normative)

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 attribute's getter must return this element's descendant text content.

The value attribute's setter must run these steps:

1. Set this element's default value override to its default value.
2. String replace all with the given value within this element.

The defaultValue attribute's getter must return the result of running this element's default value.

The defaultValue attribute's setter must run these steps:

1. If this element's default value override is null, then string replace all with the given value within this element and return.
2. Set this element's default value override to the given value.

The type attribute's getter must return "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

A simple calculator could use output for its display of calculated results:

```html
<form onsubmit="return false" oninput="o.value = a.valueAsNumber + b.valueAsNumber">
  <input id=a type=number step=any> +
  <input id=b type=number step=any> =
  <output id=o for="a b"></output>
</form>
```

Example

In this example, an output element is used to report the results of a calculation performed by a remote server, as they come in:

```html
<output id="result"></output>
<script>
  var primeSource = new WebSocket('ws://primes.example.net/');
  primeSource.onmessage = function (event) {
    document.getElementById('result').value = event.data;
  }
</script>
```

4.10.13 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

Accessibility considerations:
- For authors.
- For implementers.

DOM interface:
```idl
[Exposed=Window]
interface HTMLProgressElement : HTMLElement {
  [HTMLConstructor] constructor();

  [CEReactions] attribute double value;
  [CEReactions] attribute double max;
  readonly attribute double position;
  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 attribute specifies how much of the task has been completed, and the max 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**
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.

**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, then the **current value** is the **maximum value**, if `value` is greater than the **maximum value**, and `value` otherwise.

**UA requirements for showing the progress bar**: When representing a `progress` element to the user, the UA 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**.

**For web developers (non-normative)**

```
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.

### 4.10.14 The meter element

**Categories**: Flow content, Phrasing content, Labelable element.
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

Accessibility considerations:

For authors.

For implementers.

DOM interface:

```idl
[Exposed=Window]
interface HTMLMeterElement : HTMLElement {
    [HTMLConstructor] constructor();

    [CEReactions] attribute double value;
    [CEReactions] attribute double min;
    [CEReactions] attribute double max;
    [CEReactions] attribute double low;
    [CEReactions] attribute double high;
    [CEReactions] attribute double optimum;
    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.

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 \textit{value} be the \textit{value} attribute's number.

If the \textit{min} attribute is specified, then let \textit{minimum} be that attribute's value; otherwise, let it be zero.

If the \textit{max} attribute is specified, then let \textit{maximum} be that attribute's value; otherwise, let it be 1.0.

The following inequalities must hold, as applicable:

- \textit{minimum} \leq \textit{value} \leq \textit{maximum}
- \textit{minimum} \leq \textit{low} \leq \textit{maximum} (if \textit{low} is specified)
- \textit{minimum} \leq \textit{high} \leq \textit{maximum} (if \textit{high} is specified)
- \textit{minimum} \leq \textit{optimum} \leq \textit{maximum} (if \textit{optimum} is specified)
- \textit{low} \leq \textit{high} (if both \textit{low} and \textit{high} are specified)

\textbf{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 \texttt{meter} element.

When used with \texttt{microdata}, the \texttt{meter} element's \textit{value} attribute provides the element's machine-readable value.

\textbf{Example}

The following examples show three gauges that would all be three-quarters full:

Storage space usage: \texttt{<meter value=6 max=8>6 blocks used (out of 8 total)</meter>}
Voter turnout: \texttt{<meter value=.75><img alt="75%" src="graph75.png"/></meter>}
Tickets sold: \texttt{<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):

\texttt{<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:

\texttt{<p>The grapefruit pie had a radius of 12cm and a height of 2cm.</p>}
\texttt{<dl>}
\texttt{<dt>Radius: <dd> <meter min=0 max=20 value=12>12cm</meter>}
\texttt{<dt>Height: <dd> <meter min=0 max=10 value=2>2cm</meter>}
\texttt{</dl>}

There is no explicit way to specify units in the \texttt{meter} element, but the units may be specified in the \textit{title} attribute in free-form text.

\textbf{Example}

The example above could be extended to mention the units:

\texttt{<dl>}
\texttt{<dt>Radius: <dd> <meter min=0 max=20 value=12 title="centimeters">12cm</meter>}
\texttt{<dt>Height: <dd> <meter min=0 max=10 value=2 title="centimeters">2cm</meter>}
\texttt{</dl>}

\textbf{User agent requirements:} User agents must parse the \texttt{min}, \texttt{max}, \texttt{value}, \texttt{low}, \texttt{high}, and \texttt{optimum} attributes using the \texttt{rules for parsing floating-point number values}. 559
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 ≤ actual value ≤ maximum value`
- `minimum value ≤ low boundary ≤ high boundary ≤ maximum value`
- `minimum value ≤ optimum point ≤ maximum value`

**UA 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.

**UA requirements for showing the gauge:** When representing a `meter` element to the user, the UA 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.

Example

The following markup:

```html
<h3>Suggested groups</h3>

```

 might be rendered as follows:

**Suggested groups** - [_hide_suggested_groups](#)

- **comp.infosystems.www.authoring.stylesheets** - [join](#)
  - Group description: Layout/presentation on the [WWW](#)
  - Usenet, 618 subscribers

- **netscape.public.mozilla.xpinstall** - [join](#)
  - Group description: Mozilla XPInstall discussion.
  - Usenet, 22 subscribers

- **mozilla.dev.general** - [join](#)
  - Group description: Mozilla XPInstall discussion.
  - Usenet, 66 subscribers

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

For example, the following snippet:

```html
<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
The `value` IDL attribute, on getting, must return the `actual value`\(^{550}\). On setting, the given value must be converted to the `best representation of the number as a floating-point number`\(^{570}\) and then the `value`\(^{558}\) content attribute must be set to that string.

The `min` IDL attribute, on getting, must return the `minimum value`\(^{550}\). On setting, the given value must be converted to the `best representation of the number as a floating-point number`\(^{570}\) and then the `min`\(^{558}\) content attribute must be set to that string.

The `max` IDL attribute, on getting, must return the `maximum value`\(^{550}\). On setting, the given value must be converted to the `best representation of the number as a floating-point number`\(^{570}\) and then the `max`\(^{558}\) content attribute must be set to that string.

The `low` IDL attribute, on getting, must return the `low boundary`\(^{550}\). On setting, the given value must be converted to the `best representation of the number as a floating-point number`\(^{570}\) and then the `low`\(^{558}\) content attribute must be set to that string.

The `high` IDL attribute, on getting, must return the `high boundary`\(^{550}\). On setting, the given value must be converted to the `best representation of the number as a floating-point number`\(^{570}\) and then the `high`\(^{558}\) content attribute must be set to that string.

The `optimum` IDL attribute, on getting, must return the `optimum value`\(^{550}\). On setting, the given value must be converted to the `best representation of the number as a floating-point number`\(^{570}\) and then the `optimum`\(^{558}\) content attribute must be set to that string.

The `labels`\(^{492}\) IDL attribute provides a list of the element's `label`\(^{490}\)s.

**Example**

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>170261928 bytes used
out of 233257824 bytes available</meter></p>
```

### 4.10.15 The `fieldset` element\(^{556}\)

**Categories**\(^{129}\):
- Flow content\(^{132}\)
- Sectioning root\(^{520}\)
- Listed\(^{486}\) and autocapitalize-inheriting\(^{486}\) form-associated element\(^{486}\)
- Palpable content\(^{134}\)

**Contexts in which this element can be used**\(^{129}\):
Where flow content\(^{132}\) is expected.

**Content model**\(^{129}\):
Optionally a `legend`\(^{565}\) element, followed by flow content\(^{132}\).

**Tag omission in text/html**\(^{130}\):
Neither tag is omissible.

**Content attributes**\(^{130}\):
- Global attributes\(^{137}\)
  - `disabled`\(^{553}\) — Whether the descendant form controls, except any inside legend\(^{556}\), are disabled
  - `form`\(^{546}\) — Associates the element with a form\(^{486}\) element
  - `name`\(^{568}\) — Name of the element to use in the form.elements\(^{548}\) API.

**Accessibility considerations**\(^{130}\):
- For authors.
- For implementers.

**DOM interface**\(^{130}\):

```
IDL [Exposed=Window]
interface HTMLFieldSetElement : HTMLElement {
```

562
The **fieldset** element represents a set of form controls (or other content) grouped together, optionally with a caption. The caption 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.

**For web developers (non-normative)**

**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**

This example shows a **fieldset** element being used to group a set of related controls:

```html
<fieldset>
  <legend>Display</legend>
  <p><label><input type=radio name=c value=0 checked> Black on White</label>
  <label><input type=radio name=c value=1> White on Black</label>
</fieldset>
```
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 controls and an optional year/month control.

```
<fieldset name="clubfields" disabled>
  <legend>
    <input type=checkbox name=club onchange="form.clubfields.disabled = !checked"> Use Club Card
  </legend>
  <p>Name on card: <input name=clubname required></p>
  <p>Card number: <input name=clubnum required pattern="\[-0-9]+"></p>
  <p>Expiry date: <input name=clubexp type=month></p>
</fieldset>
```

Example

You can also nest **fieldset** elements. Here is an example expanding on the previous one that does so:

```
<fieldset name="clubfields" disabled>
  <legend>
    <input type=checkbox name=club onchange="form.clubfields.disabled = !checked"> Use Club Card
  </legend>
  <p>Name on card: <input name=clubname required></p>
  <fieldset name="numfields">
    <legend>
      <input type=radio checked name=clubtype onchange="form.numfields.disabled = !checked"> My card has numbers on it
    </legend>
    <p>Card number: <input name=clubnum required pattern="\[-0-9]+"></p>
  </fieldset>
</fieldset>
```

Example

In this example, if the outer "Use Club Card" checkbox is not checked, everything inside the outer **fieldset**s, 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.

```
<fieldset name="letfields" disabled>
  <legend>
    <input type=radio name=clubtype onchange="form.letfields.disabled = !checked"> My card has letters on it
  </legend>
  <p>Card code: <input name=clublet required pattern="[A-Za-z]+"></p>
</fieldset>
```

Example

This example shows a grouping of controls where the **legend** element both labels the grouping, and the nested heading element surfaces the grouping in the document outline:

```
<fieldset>
  <legend>
```

564
<legend> <h2>
How can we best reach you?
</h2> </legend>
<p> <label>
<input type=radio checked name=contact_pref>
Phone
</label>
</p>
<p> <label>
<input type=radio name=contact_pref>
Text
</label>
</p>
<p> <label>
<input type=radio name=contact_pref>
Email
</label>
</p>"
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.

input and textarea elements have a dirty value flag. This is used to track the interaction between the value and default value. If it is false, value mirrors the default value. If it is true, the default value is ignored.

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.

To define the behavior of the maxlength and minlength attributes, as well as other APIs specific to the textarea element, all form control with a value also have an algorithm for obtaining an API value. By default this algorithm is to simply return the control's value.

The select element does not have a value; the selectedness of its option elements is what is used instead.

A form control can be designated as mutable.

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.

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 has an associated parser inserted flag.

A form-associated element is, by default, associated with its nearest ancestor form element (as described below), but, if it is listed, may have a form attribute specified to override this.

If a listed 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 tree.
Note

The rules in this section are complicated by the fact that although conforming documents or trees will never contain nested form elements, it is quite possible (e.g., using a script that performs DOM manipulation) to generate trees 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:
1. If the form-associated element's parser inserted flag is set, then return.
2. Reset the form owner of the form-associated element.

When a form-associated element or one of its ancestors is removed, then:
1. If the form-associated element has a form and the form and its form owner are no longer in the same tree, then reset the form owner of the form-associated element.

When a listed 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 listed form-associated element has a form attribute and the ID of any of the elements in the tree changes, then the user agent must reset the form owner of that form-associated element.

When a listed 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. Unset element's parser inserted flag.
2. If all of the following conditions are true
   - element's form owner is not null
   - element is not listed, or its form content attribute is not present
   - element's form owner is its nearest form element ancestor after the change to the ancestor chain
   then do nothing, and return.
3. Set element's form owner to null.
4. If element is listed, has a form content attribute, and is connected, then:
   1. If the first element in element's tree, in tree order, to have an ID that is identical to element's form content attribute's value, is a form element, then associate the element with that form element.
5. Otherwise, if element has an ancestor form element, then associate element with the nearest such ancestor form element.

Example

In the following non-conforming snippet:

```
...<form id="a">
  <div id="b"></div>
</form>
<script>
  document.getElementById('b').innerHTML = '
    <table><tr><td></form><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, and there is a parse error for the </form> tag.

**For web developers (non-normative)**

```
let form = document.createElement("form");
let input = document.createElement("input");
form.appendChild(input);
form.method; // => "get"
input.name = "method"; // DOM clobbering occurs here
form.method === input; // => true
```

Since the input name takes precedence over built-in form properties, the JavaScript reference form.method will point to the input element named "method" instead of the built-in method property.

---

Listed except for form-associated custom elements have a form IDL attribute, which, on getting, must return the element's form owner, or null if there isn't one.

Form-associated custom elements don't have form IDL attribute. Instead, their ElementInternals object has a form IDL attribute. On getting, it must throw a "NotSupportedException" DOMException if the target element is not a form-associated custom element. Otherwise, it must return the element's form owner, or null if there isn't one.

### 4.10.18 Attributes common to form controls

#### 4.10.18.1 Naming form controls: the name attribute

The name content attribute gives the name of the form control, as used in form submission and in the form element's elements object. If the attribute is specified, its value must not be the empty string or isindex.

**Note**

A number of user agents historically implemented special support for first-in-form text controls with the name isindex, and this specification previously defined related user agent requirements for it. However, some user agents subsequently dropped that special support, and the related requirements were removed from this specification. So, to avoid problematic reinterpretations in legacy user agents, the name isindex is no longer allowed.

Other than isindex, any non-empty value for name is allowed. An ASCII case-insensitive match for the name _charset_ is special: if used as the name of a Hidden control with no value attribute, then during submission the value attribute is automatically given a value consisting of the submission character encoding.

The name IDL attribute must reflect the name content attribute.

**Note**

DOM clobbering is a common cause of security issues. Avoid using the names of built-in form properties with the name content attribute.

In this example, the input element overrides the built-in method property:

```
let form = document.createElement("form");
let input = document.createElement("input");
form.appendChild(input);

form.method; // => "get"
input.name = "method"; // DOM clobbering occurs here
form.method === input; // => true
```

Since the input name takes precedence over built-in form properties, the JavaScript reference form.method will point to the input element named "method" instead of the built-in method property.
### 4.10.18.2 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 control that contains this value during form submission. If such an attribute is specified, its value must not be the empty string.

#### Example

In this example, a form contains a text control and a submission button:

```html
<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=rtl&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
```

### 4.10.18.3 Limiting user input length: the `maxlength` attribute

A form control `maxlength` attribute, controlled by the `dirty value flag` declares a limit on the number of characters a user can input. The "number of characters" is measured using `length` and, in the case of `textarea` elements, with all newlines normalized to a single character (as opposed to CRLF pairs).

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 `length` of the element's `API 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 `API value` to be set to a value whose `length` is greater than the element's maximum allowed value length.

#### Note

In the case of `textarea` elements, the `API value` and `value` differ. In particular, newline normalization is applied before the maximum allowed value length is checked (whereas the `textarea wrapping transformation` is not applied).

### 4.10.18.4 Setting minimum input length requirements: the `minlength` attribute

A form control `minlength` attribute, controlled by the `dirty value flag`, declares a lower bound on the number of characters a user can input. The "number of characters" is measured using `length` and, in the case of `textarea` elements, with all newlines normalized to a single character (as opposed to CRLF pairs).

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

**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 **length** of the element’s API value is less than the element’s **minimum allowed value length**, then the element is **suffering from being too short**.

**Example**

In this example, there are four text controls. 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.

```html
<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>
```

### 4.10.18.5 Enabling and disabling form controls: the **disabled** attribute

The **disabled** content attribute is a **boolean attribute**.

**Note**

The **disabled** attribute for **option** elements and the **disabled** attribute for **optgroup** elements are defined separately.

A form control is **disabled** if any of the following conditions are met:

1. The element is a **button**, **input**, **select**, **textarea**, or **form-associated custom 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.

### 4.10.18.6 Form submission attributes

#### Attributes for 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
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 keyword dialog, mapping to the state dialog, indicating that submitting the form is intended to close the dialog box in which the form finds itself, if any, and otherwise not submit.

The method attribute's invalid value default and missing value default are both the GET state.

The formmethod attribute's invalid value default is the GET state. It has no missing value default.

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
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
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>
```

Example
In this example, a form is used with a dialog. The method attribute's "dialog" keyword is used to have the dialog automatically close when the form is submitted.

```
<dialog id="ship">
  <form method="dialog">
    <p>A ship has arrived in the harbour.</p>
    <button type=submit value="board">Board the ship</button>
    <button type=submit value="call">Call to the captain</button>
  </form>
</dialog>
<script>
  var ship = document.getElementById('ship');
  ship.showModal();
  ship.onclose = function (event) {
    if (ship.returnValue == 'board') {
```
The `enctype` and `formenctype` content attributes are enumerated attributes\(^68\) 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 `enctype`\(^572\) attribute's invalid value default\(^68\) and missing value default\(^68\) are both the application/x-www-form-urlencoded\(^68\) state.

The `formenctype`\(^572\) attribute's invalid value default\(^68\) is the application/x-www-form-urlencoded\(^572\) state. It has no missing value default\(^68\).

The `enctype` of an element is one of those three states. If the element is a submit button\(^486\) and has a formenctype\(^572\) attribute, then the element's `enctype`\(^572\) is that attribute's state; otherwise, it is the form owner\(^566\)'s `enctype`\(^572\) attribute's state.

The `target` and `formtarget` content attributes, if specified, must have values that are valid browsing context names or keywords\(^822\).

The `novalidate` and `formnovalidate` content attributes are boolean attributes\(^567\). If present, they indicate that the form is not to be validated during submission.

The `novalidate state` of an element is true if the element is a submit button\(^486\) and the element's `formnovalidate`\(^572\) attribute is present, or if the element's form owner\(^566\)'s `novalidate`\(^572\) attribute is present, and false otherwise.

### Example

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.

```html
<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>
  <input type=submit name=submit value="Submit essay"></input>
  <input type=submit formnovalidate name=save value="Save essay"></input>
  <input type=submit formnovalidate name=cancel value="Cancel"></input>
</form>
```

The `action` IDL attribute must reflect\(^995\) 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 element's node document's URL must be returned instead. The `target` IDL attribute must reflect\(^995\) the content attribute of the same name. The method and `enctype` IDL attributes must reflect\(^995\) the respective content attributes of the same name, limited to only known values\(^995\). The `encoding` IDL attribute must reflect\(^995\) the `enctype`\(^572\) content attribute, limited to only known values\(^995\). The `noValidate` IDL attribute must reflect\(^995\) the `novalidate`\(^572\) content attribute. The `formAction` IDL attribute must reflect\(^995\) the `formAction`\(^572\) content attribute, except that on getting, when the content attribute is missing or its value is the empty string, the element's node document's URL must be returned instead. The `formEncType` IDL attribute must reflect\(^995\) the `formenctype`\(^572\) content attribute, limited to only known values\(^995\). The `formMethod` IDL attribute must reflect\(^995\) the `formMethod`\(^572\) content attribute, limited to only known values\(^995\). The `formNoValidate` IDL attribute must reflect\(^995\) the `formnovalidate`\(^572\) content attribute. The `formTarget` IDL attribute must reflect\(^995\) the `formtarget`\(^572\) content attribute.
4.10.17.7 Autofill

4.10.18.7 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 `"on"`, or a single token that is an ASCII case-insensitive match for the string `"off"`, or `autocomplete 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 `autocomplete 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**

   For example, if there are two shipping addresses in the form, then they could be marked up as:

   ```html
   <fieldset>
     <legend>Ship the blue gift to...</legend>
     <p> <label> Address: </label> <textarea name=ba autocomplete="section-blue shipping street-address"></textarea> </p>
     <p> <label> City: </label> <input name=bc autocomplete="section-blue shipping address-level2"> </p>
     <p> <label> Postal Code: </label> <input name=bp autocomplete="section-blue shipping postal-code"> </p>
   </fieldset>
   <fieldset>
     <legend>Ship the red gift to...</legend>
     <p> <label> Address: </label> <textarea name=ra autocomplete="section-red shipping street-address"></textarea> </p>
     <p> <label> City: </label> <input name=rc autocomplete="section-red shipping address-level2"> </p>
     <p> <label> Postal Code: </label> <input name=rp autocomplete="section-red shipping postal-code"> </p>
   </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"
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"
   - "impp"
   - "photo"

(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 UA to prefill the value for them; 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

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.

```html
<form method="post" action="/step2.cgi">
  <input type="hidden" name="transaction-currency" value="CHF"><br>
  <input type="hidden" name="transaction-amount" value="15.00"
  autocomplete="cc-number"/>
  <p><label>Credit card number: <input type="text" inputmode="numeric" autocomplete="cc-number"></label></p>
  <p><label>Expiry Date: <input type="month" autocomplete="cc-exp"></label></p>
  <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 subparts 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-month" and "cc-exp-year"), 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.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Meaning</th>
<th>Canonical Format</th>
<th>Example</th>
<th>Control group</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;name&quot;</td>
<td>Full name</td>
<td>Free-form text, no newlines</td>
<td>Sir Timothy John Berners-Lee, OM, KBE, FRSA</td>
<td>Text&lt;sup&gt;575&lt;/sup&gt;</td>
</tr>
<tr>
<td>&quot;honorific-prefix&quot;</td>
<td>Prefix or title (e.g. &quot;Mr.&quot;, &quot;Ms.&quot;, &quot;Dr.&quot;, &quot;M(^{\text{Ch}})&quot;&quot;)</td>
<td>Free-form text, no newlines</td>
<td>Sir</td>
<td>Text&lt;sup&gt;575&lt;/sup&gt;</td>
</tr>
<tr>
<td>&quot;given-name&quot;</td>
<td>Given name (in some Western cultures, also known as the first name)</td>
<td>Free-form text, no newlines</td>
<td>Timothy</td>
<td>Text&lt;sup&gt;575&lt;/sup&gt;</td>
</tr>
<tr>
<td>&quot;additional-name&quot;</td>
<td>Additional names (in some Western cultures, also known as middle names, forenames other than the first name)</td>
<td>Free-form text, no newlines</td>
<td>John</td>
<td>Text&lt;sup&gt;575&lt;/sup&gt;</td>
</tr>
<tr>
<td>&quot;family-name&quot;</td>
<td>Family name (in some Western cultures, also known as the last name or surname)</td>
<td>Free-form text, no newlines</td>
<td>Berners-Lee</td>
<td>Text&lt;sup&gt;575&lt;/sup&gt;</td>
</tr>
<tr>
<td>&quot;honorific-suffix&quot;</td>
<td>Suffix (e.g. &quot;Jr.&quot;, &quot;B.Sc.&quot;, &quot;MBASW&quot;, &quot;II&quot;)</td>
<td>Free-form text, no newlines</td>
<td>OM, KBE, FRSA, FREng</td>
<td>Text&lt;sup&gt;575&lt;/sup&gt;</td>
</tr>
<tr>
<td>&quot;nickname&quot;</td>
<td>Nickname, screen name, handle: a typically short name used instead of the full name</td>
<td>Free-form text, no newlines</td>
<td>Tim</td>
<td>Text&lt;sup&gt;577&lt;/sup&gt;</td>
</tr>
<tr>
<td>Field name</td>
<td>Meaning</td>
<td>Canonical Format</td>
<td>Canonical Format Example</td>
<td>Control group</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------------------------</td>
<td>--------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>&quot;organization-title&quot;</td>
<td>Job title (e.g. &quot;Software Engineer&quot;, &quot;Senior Vice President&quot;, &quot;Deputy Managing Director&quot;)</td>
<td>Free-form text, no newlines</td>
<td>Professor</td>
<td>Text</td>
</tr>
<tr>
<td>&quot;username&quot;</td>
<td>A username</td>
<td>Free-form text, no newlines</td>
<td>timbl</td>
<td>Username</td>
</tr>
<tr>
<td>&quot;new-password&quot;</td>
<td>A new password (e.g. when creating an account or changing a password)</td>
<td>Free-form text, no newlines</td>
<td>GUMFXbadyr$3</td>
<td>Password</td>
</tr>
<tr>
<td>&quot;current-password&quot;</td>
<td>The current password for the account identified by the username field (e.g. when logging in)</td>
<td>Free-form text, no newlines</td>
<td>qwerty</td>
<td>Password</td>
</tr>
<tr>
<td>&quot;one-time-code&quot;</td>
<td>One-time code used for verifying user identity</td>
<td>Free-form text, no newlines</td>
<td>123456</td>
<td>Password</td>
</tr>
<tr>
<td>&quot;organization&quot;</td>
<td>Company name corresponding to the person, address, or contact information in the other fields associated with this field</td>
<td>Free-form text, no newlines</td>
<td>World Wide Web Consortium</td>
<td>Text</td>
</tr>
<tr>
<td>&quot;street-address&quot;</td>
<td>Street address (multiple lines, newlines preserved)</td>
<td>Free-form text</td>
<td>32 Vassar Street</td>
<td>MultiLine</td>
</tr>
<tr>
<td>&quot;address-line1&quot;</td>
<td>Street address (one line per field)</td>
<td>Free-form text, no newlines</td>
<td>32 Vassar Street</td>
<td>Text</td>
</tr>
<tr>
<td>&quot;address-line2&quot;</td>
<td>The third administrative level, in addresses with three or more administrative levels</td>
<td>Free-form text, no newlines</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>&quot;address-line3&quot;</td>
<td>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</td>
<td>Free-form text, no newlines</td>
<td>Cambridge</td>
<td>Text</td>
</tr>
<tr>
<td>&quot;address-line4&quot;</td>
<td>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</td>
<td>Free-form text, no newlines</td>
<td>MA</td>
<td>Text</td>
</tr>
<tr>
<td>&quot;country&quot;</td>
<td>Country code</td>
<td>Valid ISO 3166-1-alpha-2 country code</td>
<td>US</td>
<td>Text</td>
</tr>
<tr>
<td>&quot;country-name&quot;</td>
<td>Country name</td>
<td>Free-form text, no newlines</td>
<td>US</td>
<td>Text</td>
</tr>
<tr>
<td>&quot;postal-code&quot;</td>
<td>Postal code, post code, ZIP code, CEDEX code (if CEDEX, append &quot;CEDEX&quot;, and the arrondissement, if relevant, to the address-level4 field)</td>
<td>Free-form text, no newlines</td>
<td>02139</td>
<td>Text</td>
</tr>
<tr>
<td>&quot;cc-name&quot;</td>
<td>Full name as given on the payment instrument</td>
<td>Free-form text, no newlines</td>
<td>Tim Berners-Lee</td>
<td>Text</td>
</tr>
<tr>
<td>&quot;cc-given-name&quot;</td>
<td>Given name as given on the payment instrument (in some Western cultures, also known as the first name)</td>
<td>Free-form text, no newlines</td>
<td>Tim</td>
<td>Text</td>
</tr>
<tr>
<td>&quot;cc-additional-name&quot;</td>
<td>Additional names given on the payment instrument (in some Western cultures, also known as middle names, forenames other than the first name)</td>
<td>Free-form text, no newlines</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>&quot;cc-family-name&quot;</td>
<td>Family name given on the payment instrument (in some Western cultures, also known as the last name or surname)</td>
<td>Free-form text, no newlines</td>
<td>Berners-Lee</td>
<td>Text</td>
</tr>
<tr>
<td>&quot;cc-number&quot;</td>
<td>Code identifying the payment instrument (e.g. the credit card number)</td>
<td>ASCII digits</td>
<td>4114360123456785</td>
<td>Text</td>
</tr>
<tr>
<td>&quot;cc-exp&quot;</td>
<td>Expiration date of the payment instrument</td>
<td>Valid month string</td>
<td>2014-12</td>
<td>Month</td>
</tr>
<tr>
<td>&quot;cc-exp-month&quot;</td>
<td>Month component of the expiration date of the payment instrument</td>
<td>Valid integer in the range 1..12</td>
<td>12</td>
<td>Numeric</td>
</tr>
<tr>
<td>&quot;cc-exp-year&quot;</td>
<td>Year component of the expiration date of the payment instrument</td>
<td>Valid integer greater than zero</td>
<td>2014</td>
<td>Numeric</td>
</tr>
<tr>
<td>&quot;cc-csc&quot;</td>
<td>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)</td>
<td>ASCII digits</td>
<td>419</td>
<td>Text</td>
</tr>
<tr>
<td>&quot;cc-type&quot;</td>
<td>Type of payment instrument</td>
<td>Free-form text, no newlines</td>
<td>Visa</td>
<td>Text</td>
</tr>
<tr>
<td>&quot;transaction-currency&quot;</td>
<td>The currency that the user would prefer the transaction to use</td>
<td>ISO 4217 currency code</td>
<td>GBP</td>
<td>Text</td>
</tr>
<tr>
<td>&quot;transaction-amount&quot;</td>
<td>The amount that the user would like for the transaction (e.g. when entering a bid or sale price)</td>
<td>Valid floating-point number</td>
<td>401.00</td>
<td>Numeric</td>
</tr>
<tr>
<td>Field name</td>
<td>Meaning</td>
<td>Canonical Format</td>
<td>Control group</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>------------------</td>
<td>---------------</td>
<td></td>
</tr>
<tr>
<td>&quot;language&quot;</td>
<td>Preferred language</td>
<td>Valid BCP 47 language tag ([BCP47] en)</td>
<td>Text[^577]</td>
<td></td>
</tr>
<tr>
<td>&quot;bday&quot;</td>
<td>Birthday</td>
<td>Valid date string[^75]</td>
<td>Date[^578]</td>
<td></td>
</tr>
<tr>
<td>&quot;bday-day&quot;</td>
<td>Day component of birthday</td>
<td>Valid integer[^66] in the range 1..31</td>
<td>Numeric[^578]</td>
<td></td>
</tr>
<tr>
<td>&quot;bday-month&quot;</td>
<td>Month component of birthday</td>
<td>Valid integer[^66] in the range 1..12</td>
<td>Numeric[^578]</td>
<td></td>
</tr>
<tr>
<td>&quot;bday-year&quot;</td>
<td>Year component of birthday</td>
<td>Valid integer[^66] greater than zero</td>
<td>Numeric[^578]</td>
<td></td>
</tr>
<tr>
<td>&quot;sex&quot;</td>
<td>Gender identity (e.g. Female, Fa’afafine)</td>
<td>Free-form text, no newlines</td>
<td>Male[^577]</td>
<td></td>
</tr>
<tr>
<td>&quot;url&quot;</td>
<td>Home page or other web page corresponding to the company, person, address, or contact information in the other fields associated with this field</td>
<td>Valid URL string</td>
<td>URL[^578]</td>
<td></td>
</tr>
<tr>
<td>&quot;photo&quot;</td>
<td>Photograph, icon, or other image corresponding to the company, person, address, or contact information in the other fields associated with this field</td>
<td>Valid URL string</td>
<td>URL[^578]</td>
<td></td>
</tr>
<tr>
<td>&quot;tel&quot;</td>
<td>Full telephone number, including country code</td>
<td>ASCII digits and U+0020 SPACE characters, prefixed by a U+002B PLUS SIGN character (+)</td>
<td>Tel[^578]</td>
<td></td>
</tr>
<tr>
<td>&quot;tel-country-code&quot;</td>
<td>Country code component of the telephone number</td>
<td>ASCII digits prefixed by a U+002B PLUS SIGN character (+)</td>
<td>Text[^577]</td>
<td></td>
</tr>
<tr>
<td>&quot;tel-national&quot;</td>
<td>Telephone number without the country code component, with a country-internal prefix applied if applicable</td>
<td>ASCII digits and U+0020 SPACE characters</td>
<td>Text[^577]</td>
<td></td>
</tr>
<tr>
<td>&quot;tel-area-code&quot;</td>
<td>Area code component of the telephone number, with a country-internal prefix applied if applicable</td>
<td>ASCII digits</td>
<td>Text[^577]</td>
<td></td>
</tr>
<tr>
<td>&quot;tel-local&quot;</td>
<td>Telephone number without the country code and area code components</td>
<td>ASCII digits</td>
<td>Text[^577]</td>
<td></td>
</tr>
<tr>
<td>&quot;tel-local-prefix&quot;</td>
<td>First part of the component of the telephone number that follows the area code, when that component is split into two components</td>
<td>ASCII digits</td>
<td>Text[^577]</td>
<td></td>
</tr>
<tr>
<td>&quot;tel-local-suffix&quot;</td>
<td>Second part of the component of the telephone number that follows the area code, when that component is split into two components</td>
<td>ASCII digits</td>
<td>Text[^577]</td>
<td></td>
</tr>
<tr>
<td>&quot;tel-extension&quot;</td>
<td>Telephone number internal extension code</td>
<td>ASCII digits</td>
<td>Text[^577]</td>
<td></td>
</tr>
<tr>
<td>&quot;email&quot;</td>
<td>Email address</td>
<td>Valid email address[^503]</td>
<td>Username[^578]</td>
<td></td>
</tr>
<tr>
<td>&quot;impp&quot;</td>
<td>URL representing an instant messaging protocol endpoint (for example, &quot;aim:goim?screenname=example&quot; or &quot;<a href="xmpp:fred@example.net">xmpp:fred@example.net</a>&quot;)</td>
<td>Valid URL string</td>
<td>URL[^578]</td>
<td></td>
</tr>
</tbody>
</table>

The groups correspond to controls as follows:

**Text**
- input[^493] elements with a type[^495] attribute in the Hidden[^499] state
- input[^493] elements with a type[^495] attribute in the Search[^499] state
- textarea[^548] elements
- select[^537] elements

**Multiline**
- input[^493] elements with a type[^495] attribute in the Hidden[^499] state
- textarea[^548] elements
- select[^537] elements

**Password**
- input[^493] elements with a type[^495] attribute in the Hidden[^499] state
- input[^493] elements with a type[^495] attribute in the Search[^499] state
- input[^493] elements with a type[^495] attribute in the Password[^503] state
- textarea[^548] 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
textarea elements
select elements

**Username**

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

**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
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
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
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
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).

**4.10.18.7.2 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 whose information concerns the same subject, 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 ASCII whitespace.
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.

<table>
<thead>
<tr>
<th>Token</th>
<th>Maximum number of tokens</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;off&quot;</td>
<td>1</td>
<td>Off</td>
</tr>
<tr>
<td>&quot;on&quot;</td>
<td>1</td>
<td>Automatic</td>
</tr>
<tr>
<td>&quot;name&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;honorific-prefix&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;given-name&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;additional-name&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;family-name&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;honorific-suffix&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;nickname&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;organization-title&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;username&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;new-password&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;current-password&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;one-time-code&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;organization&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;street-address&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;address-line1&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;address-line2&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;address-line3&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;address-level4&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;address-level3&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;address-level2&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;address-level1&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;country&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;country-name&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;postal-code&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;cc-name&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;cc-given-name&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;cc-additional-name&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;cc-family-name&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;cc-number&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;cc-exp&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;cc-exp-month&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;cc-exp-year&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;cc-cc&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
<tr>
<td>&quot;cc-type&quot;</td>
<td>3</td>
<td>Normal</td>
</tr>
</tbody>
</table>
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, return.

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, return.

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. Return.
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 return.
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 `Email` 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.

In addition, when an element's autofill field name is "off", values are reset when traversing the history.

Example

Banks frequently do not want UAs to prefill login information:

```html
<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

For example, suppose a user visits a page with this control:

```html
<select name="country">
  <option>Afghanistan</option>
  <option>Albania</option>
  <option>Algeria</option>
  <option>Andorra</option>
  <option>Angola</option>
  <option>Antigua and Barbuda</option>
  <option>Argentina</option>
  <option>Armenia</option>
  <!-- ... -->
  <option>Yemen</option>
  <option>Zambia</option>
  <option>Zimbabwe</option>
</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 its tree, 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.

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
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 prefilled values given the aforementioned information would be "123" and "1234", or "666" and "7777", respectively.

Example
Similarly, if a form for some reason contained both a "`cc-exp`" field and a "`cc-exp-month`" field, and the user agent prefilled the form, then the month component of the former would have to match the latter.

Example
This requirement interacts with the `autofill anchor mantle` also. Consider the following markup snippet:

```html
<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 control 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

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

For example, if a select element only has option elements with values "Steve" and "Rebecca", "Jay", and "Bob", and has an autofill field 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.

Example

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:

```html
<input name="middle-initial" maxlength=1 autocomplete="additional-name">
```

...then the user agent could convert "Ines" to "I" and prefill it that way.

Example

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:

<table>
<thead>
<tr>
<th>Control</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;input name=&quot;b&quot; type=month autocomplete=&quot;bday&quot;&gt;</code></td>
<td><code>2012-07</code></td>
</tr>
<tr>
<td><code>&lt;select name=&quot;c&quot; autocomplete=&quot;bday&quot;&gt;</code></td>
<td><code>July</code></td>
</tr>
<tr>
<td><code>&lt;option&gt;Jan&lt;/option&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;option&gt;Feb&lt;/option&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;option&gt;Jul&lt;/option&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;option&gt;Aug&lt;/option&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;/select&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;input name=&quot;a&quot; type=number min=1 max=12 autocomplete=&quot;bday-month&quot;&gt;</code></td>
<td><code>7</code></td>
</tr>
<tr>
<td><code>&lt;input name=&quot;a&quot; type=number min=0 max=11 autocomplete=&quot;bday-month&quot;&gt;</code></td>
<td><code>6</code></td>
</tr>
</tbody>
</table>

The day is dropped since the Month state only accepts a month/year combination. (Note that this example is non-conforming, because the autofill field name is not allowed with the Month state.)

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.

User agent converts "July" to a month number in the range 1..12, like the field.

User agent converts "July" to a month number in the range 0..11, like the field.
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.

<table>
<thead>
<tr>
<th>Form control</th>
<th>New autofill field name</th>
</tr>
</thead>
<tbody>
<tr>
<td>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</td>
<td>username</td>
</tr>
<tr>
<td>an input element whose type attribute is in the Password state that is preceded by an input element whose autofill field name is ”username”</td>
<td>”current-password”</td>
</tr>
<tr>
<td>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”</td>
<td>new-password</td>
</tr>
<tr>
<td>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”</td>
<td>new-password</td>
</tr>
</tbody>
</table>

The autocomplete IDL attribute, on getting, must return the element’s IDL-exposed autofill value, and on setting, must reflect the content attribute of the same name.

4.10.19 APIs for the text control selections

The input and textarea elements define several attributes and methods for handling their selection. Their shared algorithms are defined here.

**For web developers (non-normative)**

**element.select()**

Selects everything in the text control.

**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 will be set after the text has been replaced. The possible values are:
All input elements to which these APIs apply, and all textarea elements, have either a selection or a text entry cursor position at all times (even for elements that are not being rendered), measured in offsets into the code units of the control's relevant value. The initial state must consist of a text entry cursor position at the beginning of the control.

For input elements, these APIs must operate on the element's value. For textarea elements, these APIs must operate on the element's API value. In the below algorithms, we call the value string being operated on the relevant value.

Whenever the relevant value changes for an element to which these APIs apply, run these steps:

1. If the element has a selection:
   1. If the start of the selection is now past the end of the relevant value, set it to the end of the relevant value.
   2. If the end of the selection is now past the end of the relevant value, set it to the end of the relevant value.
   3. If the user agent does not support empty selection, and both the start and end of the selection are now pointing to the end of the relevant value, then instead set the element's text entry cursor position to the end of the relevant value, removing any selection.

2. Otherwise, the element must have a text entry cursor position position. If it is now past the end of the relevant value, set it to the end of the relevant value.

Note

In some cases where the relevant value changes, other parts of the specification will also modify the text entry cursor position beyond just the clamping steps above. For example, see the value setter for textarea.

Where possible, user interface features for changing the text selection in input and textarea elements must be implemented using the set the selection range algorithm so that, e.g., all the same events fire.

The selections of input and textarea elements have a selection direction, which is either "forward", "backward", or "none". The exact meaning of the selection direction depends on the platform. This direction is set when the user manipulates the

Example

The use of API value instead of raw value for textarea elements means that U+000D (CR) characters are normalized away. For example,

```html
<textarea id="demo"></textarea>
<script>
  demo.value = "A\r\nB";
  demo.setRangeText("replaced", 0, 2);
  assert(demo.value === "replacedB");
</script>
```

If we had operated on the raw value of "A\r\nB", then we would have replaced the characters "A\r", ending up with a result of "replaced\nB". But since we used the API value of "A\nB", we replaced the characters "A\n", giving "replacedB".

Note

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

Whenever the relevant value changes for an element to which these APIs apply, run these steps:

1. If the element has a selection:
   1. If the start of the selection is now past the end of the relevant value, set it to the end of the relevant value.
   2. If the end of the selection is now past the end of the relevant value, set it to the end of the relevant value.
   3. If the user agent does not support empty selection, and both the start and end of the selection are now pointing to the end of the relevant value, then instead set the element's text entry cursor position to the end of the relevant value, removing any selection.

2. Otherwise, the element must have a text entry cursor position position. If it is now past the end of the relevant value, set it to the end of the relevant value.

Note

In some cases where the relevant value changes, other parts of the specification will also modify the text entry cursor position beyond just the clamping steps above. For example, see the value setter for textarea.

Where possible, user interface features for changing the text selection in input and textarea elements must be implemented using the set the selection range algorithm so that, e.g., all the same events fire.

The selections of input and textarea elements have a selection direction, which is either "forward", "backward", or "none". The exact meaning of the selection direction depends on the platform. This direction is set when the user manipulates the
selection. The initial selection direction must be "none" if the platform supports that direction, or "forward" otherwise.

To set the selection direction of an element to a given direction, update the element's selection direction to the given direction, unless the direction is "none" and the platform does not support that direction; in that case, update the element's selection to "forward".

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 "backward" 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, when invoked, must run the following steps:

1. If this element is an input element, and either select does not apply to this element or the corresponding control has no selectable text, return.

Example
For instance, in a user agent where <input type=color> is rendered as a color well with a picker, as opposed to a text control accepting a hexadecimal color code, there would be no selectable text, and thus calls to the method are ignored.

2. Set the selection range with 0 and infinity.

The selectionStart attribute's getter must run the following steps:

1. If this element is an input element, and selectionStart does not apply to this element, return null.

2. If there is no selection, return the code unit offset within the relevant value to the character that immediately follows the text entry cursor.

3. Return the code unit offset within the relevant value to the character that immediately follows the start of the selection.

The selectionStart attribute's setter must run the following steps:

1. If this element is an input element, and selectionStart does not apply to this element, throw an "InvalidStateError" DOMException.

2. Set the selection range with the value of this element's selectionStart attribute, the given value, and the value of the element's selectionDirection attribute.

The selectionEnd attribute's getter must run the following steps:

1. If this element is an input element, and selectionEnd does not apply to this element, return null.

2. If there is no selection, return the code unit offset within the relevant value to the character that immediately follows the text entry cursor.

3. Return the code unit offset within the relevant value to the character that immediately follows the end of the selection.

The selectionEnd attribute's setter must run the following steps:

1. If this element is an input element, and selectionEnd does not apply to this element, throw an "InvalidStateError" DOMException.

2. Set the selection range with the value of this element's selectionStart attribute, the given value, and the value of
this element's `selectionDirection` attribute.

The `selectionDirection` attribute's getter must run the following steps:

1. If this element is an `input` element, and `selectionDirection` does not apply to this element, return null.
2. Return this element's `selection direction`.

The `selectionDirection` attribute's setter must run the following steps:

1. If this element is an `input` element, and `selectionDirection` does not apply to this element, throw an `InvalidStateError` DOMException.
2. Set the selection range with the value of this element's `selectionStart` attribute, the value of this element's `selectionEnd` attribute, and the given value.

The `setSelectionRange(start, end, direction)` method, when invoked, must run the following steps:

1. If this element is an `input` element, and `setSelectionRange()` does not apply to this element, throw an `InvalidStateError` DOMException.
2. Set the selection range with `start`, `end`, and `direction`.

To set the selection range with an integer or null `start`, an integer or null or the special value infinity `end`, and optionally a string `direction`, run the following steps:

1. If `start` is null, let `start` be zero.
2. If `end` is null, let `end` be zero.
3. Set the `selection` of the text control to the sequence of `code units` within the `relevant value` starting with the code unit at the start position (in logical order) and ending with the code unit at the (end-1)th position. Arguments greater than the `length` of the `relevant value` of the text control (including the special value infinity) must be treated as pointing at the end of the text control. 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 UAs where there is no concept of an empty selection, this must set the cursor to be just before the character with offset `end`.
4. If `direction` is not identical to either "backward" or "forward", or if the `direction` argument was not given, set `direction` to "none".
5. Set the selection direction of the text control to `direction`.
6. If the previous steps caused the `selection` of the text control to be modified (in either extent or `direction`), then queue an element task on the user interaction task source given the element to fire an event named `select` at the element, with the `bubbles` attribute initialized to true.

The `setRangeText(replacement, start, end, selectMode)` method, when invoked, must run the following steps:

1. If this element is an `input` element, and `setRangeText()` does not apply to this element, throw an `InvalidStateError` DOMException.
2. Set this element's `dirty value flag` to true.
3. If the method has only one argument, then let `start` and `end` have the values of the `selectionStart` attribute and the `selectionEnd` attribute respectively. Otherwise, let `start`, `end` have the values of the second and third arguments respectively.
4. If `start` is greater than `end`, then throw an `IndexSizeError` DOMException.
5. If `start` is greater than the `length` of the `relevant value` of the text control, then set it to the `length` of the `relevant value` of the text control.
6. If `end` is greater than the `length` of the `relevant value` of the text control, then set it to the `length` of the `relevant value` of the text control.
7. Let `selection start` be the current value of the `selectionStart` attribute.
8. Let `selection end` be the current value of the `selectionEnd` attribute.
9. If start is less than end, delete the sequence of code units within the element's relevant value starting with the code unit at the start position and ending with the code unit at the (end-1)th position.

10. Insert the value of the first argument into the text of the relevant value of the text control, immediately before the start code unit.

11. Let new length be the length of the value of the first argument.

12. Let new end be the sum of start and new length.

13. 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"
   ↪ If the method has only one argument
      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.)

14. Set the selection range with selection start and selection end.

The setRangeText() method uses the following enumeration:

```
enum SelectionMode {
  "select",
  "start",
  "end",
  "preserve" // default
};
```

Example
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
To add some text at the start of a text control, while maintaining the text selection, the three attributes must be preserved:
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 has a datalist element ancestor.)

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, except for form-associated custom elements. Form-associated custom elements can have a custom validity error message set via their `ElementInternals` object's `setValidity()` 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, textarea) or, more complicated rules for select and controls in radio button groups, as specified in their sections.

When the `setValidity()` method sets valueMissing flag to true for a form-associated custom element.

**Suffering from a type mismatch**
When a control that allows arbitrary user input has a value that is not in the correct syntax (Email, URL).

When the `setValidity()` method sets typeMismatch flag to true for a form-associated custom element.

**Suffering from a pattern mismatch**
When a control has a value that doesn't satisfy the pattern attribute.

When the `setValidity()` method sets patternMismatch flag to true for a form-associated custom element.

**Suffering from being too long**
When a control has a value that is too long for the form control `maxlength` attribute (input, textarea).

When the `setValidity()` method sets tooLong flag to true for a form-associated custom element.

**Suffering from being too short**
When a control has a value that is too short for the form control `minlength` attribute (input, textarea).

When the `setValidity()` method sets tooShort flag to true for a form-associated custom element.

**Suffering from an underflow**
When a control has a value that is not the empty string and is too low for the `min` attribute.

When the `setValidity()` method sets rangeUnderflow flag to true for a form-associated custom element.
Suffering from an overflow
When a control has a `value` that is not the empty string and is too high for the `max` attribute.

When the `setValidity()` method sets `rangeOverflow` flag to true for a form-associated custom element.

Suffering from a step mismatch
When a control has a `value` that doesn't fit the rules given by the `step` attribute.

When the `setValidity()` method sets `stepMismatch` flag to true for a form-associated custom element.

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.

When the `setValidity()` method sets `badInput` flag to true for a form-associated custom element.

Suffering from a custom error
When a control's custom validity error message (as set by the element's `setCustomValidity()` method or `ElementInternals`'s `setValidity()` method) is not the empty string.

An element satisfies its constraints if it is not suffering from any of the above validity states.

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.

4.10.20.2 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:
   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.
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:
   1. Let `notCanceled` be the result of firing an event named `invalid` at `field`, with the `cancelable` attribute initialized to true.
   2. If `notCanceled` is true, 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.
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. For elements that are form-associated custom elements, user agents should use their validation anchor instead, for the purposes of these actions.

   ◦ 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.

4.10.20.3 The constraint validation API

For web developers (non-normative)

```javascript
// in the element

element.willValidate
  Returns true if the element will be validated when the form is submitted; false otherwise.

element.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.
```
The `willValidate` attribute's getter must return true, if this element is a candidate for constraint validation, and false otherwise (i.e., false if any conditions are barring it from constraint validation, and if the event isn’t canceled) reports the problem to the user.

The `willValidate` attribute of `ElementInternals` interface, on getting, must throw a "NotSupportedError" DOMException if the target element is not a form-associated custom element. Otherwise, it must return true if the target element is a candidate for constraint validation, and false otherwise.

The `setCustomValidity(message)` method, when invoked, must set the custom validity error message to `message`.

The `validity` attribute's getter must return a `ValidityState` object that represents the validity states of this element. This object is live.

The `validity` attribute of `ElementInternals` interface, on getting, must throw a "NotSupportedError" DOMException if the target element is not a form-associated custom element. Otherwise, it must return a `ValidityState` object that represents the validity states of the target element. This object is live.

#### IDL

```
[Exposed=Window]
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.

---

**Example**

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.

```html
<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` attribute's getter must return a `ValidityState` object that represents the validity states of this element. This object is live.

The `validity` attribute of `ElementInternals` interface, on getting, must throw a "NotSupportedError" DOMException if the target element is not a form-associated custom element. Otherwise, it must return a `ValidityState` object that represents the validity states of the target element. This object is live.
valueMissing
   The control is suffering from being missing\(^{590}\).

typeMismatch
   The control is suffering from a type mismatch\(^{590}\).

patternMismatch
   The control is suffering from a pattern mismatch\(^{590}\).

tooLong
   The control is suffering from being too long\(^{590}\).

tooShort
   The control is suffering from being too short\(^{590}\).

rangeUnderflow
   The control is suffering from an underflow\(^{590}\).

rangeOverflow
   The control is suffering from an overflow\(^{590}\).

stepMismatch
   The control is suffering from a step mismatch\(^{591}\).

badInput
   The control is suffering from bad input\(^{591}\).

customError
   The control is suffering from a custom error\(^{591}\).

valid
   None of the other conditions are true.

The check validity steps for an element \(element\) are:

1. If \(element\) is a candidate for constraint validation\(^{590}\) and does not satisfy its constraints\(^{591}\), then:
   1. Fire an event named invalid\(^{1266}\) at \(element\), with the cancelable attribute initialized to true (though canceling has no effect).
   2. Return false.
2. Return true.

The checkValidity() method, when invoked, must run the check validity steps\(^{594}\) on this element.

The checkValidity() method of the ElementInternals\(^{716}\) interface must run these steps:

1. Let \(element\) be this ElementInternals\(^{716}\)’s target element\(^{716}\).
2. If \(element\) is not a form-associated custom element\(^{705}\), then throw a "NotSupportedException" DOMException.
3. Run the check validity steps\(^{594}\) on \(element\).

The report validity steps for an element \(element\) are:

1. If \(element\) is a candidate for constraint validation\(^{590}\) and does not satisfy its constraints\(^{591}\), then:
   1. Let \(report\) be the result of firing an event named invalid\(^{1266}\) at \(element\), with the cancelable attribute initialized to true.
   2. If \(report\) is true, then report the problems with the constraints of this element to the user. When reporting the problem with the constraints to the user, the user agent may run the focusing steps\(^{718}\) for \(element\), and may change the scrolling position of the document, or perform some other action that brings \(element\) to the user’s attention. User agents may report more than one constraint violation, if \(element\) suffers from multiple problems at
once. If element is not being rendered, then the user agent may, instead of notifying the user, report the error for the running script.

3. Return false.

The reportValidity() method, when invoked, must run the report validity steps on this element.

The reportValidity() method of the ElementInternals interface must run these steps:

1. Let element be this ElementInternals's target element.
2. If element is not a form-associated custom element, then throw a "NotSupportedException" DOMException.
3. Run the report validity steps on element.

The validationMessage attribute's getter must run these steps:

1. If this element is not a candidate for constraint validation or if this element satisfies its constraints, then return the empty string.
2. 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 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.

4.10.20.4 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.

4.10.21 Form submission

4.10.21.1 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:

```html
<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:

```html
<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:

```
------kYFrd4jNJEGCervE
Content-Disposition: form-data; name="t"
cats
------kYFrd4jNJEGCervE
Content-Disposition: form-data; name="q"
fur
------kYFrd4jNJEGCervE--
```

### 4.10.21.2 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 control is focused implicitly submits the form), then doing so for a form, whose default button has activation behavior and is not disabled, must cause the user agent to fire a click event at that default button.

> 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 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, Email, Password, Date, Month, Week, Time, Local Date and Time, Number.

### 4.10.21.3 Form submission algorithm

Each form element has a constructing entry list boolean, initially false.

Each form element has a firing submission events boolean, initially false.

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. If form cannot navigate, then return.
2. If form's constructing entry list is true, then return.
4. If form document's active sandboxing flag set has its sandboxed forms browsing context flag set, then return.
5. Let form browsing context be the browsing context of form document.
6. If the submitted from submit() method flag is not set, then:
   1. If form's firing submission events is true, then return.
   2. Set form's firing submission events to true.
   3. If 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 (i.e., the constraint validation concluded that there were invalid fields and probably informed the user of this), then:
      1. Set form's firing submission events to false.
2. Return.

4. Let submitterButton be null if submitter is form. Otherwise, let submitterButton be submitter.

5. Let continue be the result of firing an event named submit at form using SubmitEvent, with the submitter attribute initialized to submitterButton, the bubbles attribute initialized to true, and the cancelable attribute initialized to true.

6. Set form's firing submission events to false.

7. If continue is false, then return.

8. If form cannot navigate, then return.

9. Let encoding be the result of picking an encoding for the form.

10. Let entry list be the result of constructing the entry list with form, submitter, and encoding.

11. If form cannot navigate, then return.

12. Let action be the submitter element's action.

13. If action is the empty string, let action be the URL of the form document.

14. Parse a URL given action, relative to the submitter element's node document. If this fails, return.

15. Let scheme be the scheme of parsed action.

16. Let enctype be the submitter element's enctype.

17. Let method be the submitter element's method.

18. Let target be the submitter element's formtarget attribute value, if the element is a submit button and has such an attribute. Otherwise, let it be the result of getting an element's target given submitter's form owner.

19. Let noopener be the result of getting an element's noopener with form and targetAttributeValue.

20. Let historyHandling be "replace" if windowType is either "new and unrestricted" or "new with no opener"; otherwise "default".

21. If target browsing context is null, then return.

22. If form document has not yet completely loaded and the submitted from submit() method flag is set, then set historyHandling to "replace".

23. If the value of method is dialog, then jump to the submit dialog steps. 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.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>GET</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>http</td>
<td>Mutate action URL</td>
<td>Submit as entity body</td>
</tr>
<tr>
<td>https</td>
<td>Mutate action URL</td>
<td>Submit as entity body</td>
</tr>
<tr>
<td>ftp</td>
<td>Get action URL</td>
<td>Get action URL</td>
</tr>
<tr>
<td>javascript</td>
<td>Get action URL</td>
<td>Get action URL</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td>Mutate action URL[^18]</td>
<td>Get action URL[^18]</td>
</tr>
<tr>
<td>mailto</td>
<td>Mail with headers[^15]</td>
<td>Mail as body[^19]</td>
</tr>
</tbody>
</table>

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[^31]`; when the `form` is first created, its `planned navigation`[^58] 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 `destination` is not a `request`, then set `destination` to a new `request` whose `URL` is `destination`.
2. If the `form` element’s `link types`[^295] include the `noreferrer`[^305] keyword, then set `destination`'s `referrer` to "no-referrer".
3. If the `form` has a non-null `planned navigation`, remove it from its `task queue`[^931].
4. Queue an element `task`[^932], on the DOM manipulation `task source`[^938] given the `form` element and the following steps:
   1. Set the `form`'s `planned navigation` to null.
   2. Navigate[^875] target browsing context to `destination`, with `historyHandling`[^875] set to `historyHandling` and `navigationType`[^875] set to "form-submission".
5. Set the `form`'s `planned navigation` to the just-queued `task`[^931].

The behaviors are as follows:

**Mutate action URL**

Let `query` be the result of running the `application/x-www-form-urlencoded` serializer with `entry list` and `encoding`.

Set `parsed action`'s `query` component to `query`.

**Plan to navigate**[^58] to `parsed action`.

**Submit as entity body**

Switch on `enctype`:

- **application/x-www-form-urlencoded**[^672]
  
  Let `body` be the result of running the `application/x-www-form-urlencoded` serializer with `entry list` and `encoding`.

  Set `body` to the result of `encoding` `body`.

  Let `MIME type` be "application/x-www-form-urlencoded".

- **multipart/form-data**[^672]
  
  Let `body` be the result of running the `multipart/form-data encoding algorithm`[^660] with `entry list` and `encoding`.

  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`[^662] generated by the `multipart/form-data encoding algorithm`[^662].

- **text/plain**[^672]
  
  Let `body` be the result of running the `text/plain encoding algorithm`[^660] with `entry list`.

  Set `body` to the result of `encoding` `body` using `encoding`.

  Let `MIME type` be "text/plain".

**Plan to navigate**[^58] to a new `request` whose `url` is `parsed action`, `method` is `method`, `header list` consists of `"Content-Type":MIME type`, and `body` is `body`.

**Get action URL**

**Plan to navigate**[^58] to `parsed action`.
Mail with headers

Let headers be the result of running the `application/x-www-form-urlencoded serializer` with entry list and encoding.

Replace occurrences of U+002B PLUS SIGN characters (+) in headers with the string "%20".

Set parsed action's query to headers.

Plan to navigate\(^{598}\) to parsed action.

Mail as body

Switch on `enctype`:

\[\rightarrow text/plain^{672}\]

Let body be the result of running the `text/plain encoding algorithm`\(^{662}\) with entry list.

Set body to the result of running UTF-8 percent-encode on body using the default encode set. [URL]\(^{1277}\)

\[\rightarrow Otherwise\]

Let body be the result of running the `application/x-www-form-urlencoded serializer` with entry list and encoding.

If parsed action's query is null, then set it to the empty string.

If parsed action's query is not the empty string, then append a single U+0026 AMPERSAND character (&) to it.

Append "body=" to parsed action's query.

Append body to parsed action's query.

Plan to navigate\(^{598}\) to parsed action.

Submit dialog

Let subject be the nearest ancestor `dialog`\(^{638}\) element of form, if any.

If there isn't one, or if it does not have an open\(^{611}\) attribute, do nothing. Otherwise, proceed as follows:

If submitter is an `input`\(^{493}\) element whose type\(^{495}\) attribute is in the Image Button\(^{618}\) state, then let result be the string formed by concatenating the selected coordinate\(^{519}\)'s x-component, expressed as a base-ten number using ASCII digits, a U+002C COMMA character (,), and the selected coordinate\(^{519}\)'s y-component, expressed in the same way as the x-component.

Otherwise, if submitter has a value\(^{566}\), then let result be that value\(^{566}\).

Otherwise, there is no result.

Then, close the dialog\(^{612}\) subject. If there is a result, let that be the return value.

4.10.21.4 Constructing the entry list\(^{559}\)

The algorithm to construct the entry list given a form, an optional submitter, and an optional encoding, is as follows. If not specified otherwise, submitter is null.

1. If form's constructing entry list\(^{556}\) is true, then return null.
2. Set form's constructing entry list\(^{556}\) to true.
3. Let controls be a list of all the submittable elements\(^{546}\) whose form owner\(^{556}\) is form, in tree order.
4. Let entry list be a new empty list of entries.
5. For each element field in controls, in tree order:
1. If any of the following is true:
   - 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.
   
   Then continue.

2. If the field element is an input element whose type attribute is in the Image Button state, then:
   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 entry list with name\(_x\) and x.
   6. Append an entry to entry list with name\(_y\) and y.
   7. Continue.

3. If the field is a form-associated custom element, then perform the entry construction algorithm given field and entry list, then continue.

4. If either the field element does not have a name attribute specified, or its name attribute's value is the empty string, then continue.

5. Let name be the value of the field element's name attribute.

6. 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 entry list with name and the value of the option element.

7. Otherwise, if the field element is an input element whose type attribute is in the Checkbox state or the Radio Button state, then:
   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 entry list with name and value.

8. Otherwise, if the field element is an input element whose type attribute is in the File Upload state, then:
   1. If there are no selected files, then append an entry to entry list with name and a new File object with an empty name, application/octet-stream as type, and an empty body.
   2. Otherwise, for each file in selected files, append an entry to entry list with name and a File object representing the file.

9. Otherwise, if the field element is an input element whose type attribute is in the Hidden state and name is an ASCII case-insensitive match for "_charset_":
   1. Let charset be the name of encoding if encoding is given, and "UTF-8" otherwise.
2. Append an entry to entry list with name and charset.

10. Otherwise, if the field element is a textarea element, append an entry to entry list with name and the value of the field element, and the prevent line break normalization flag set.

Note
In the case of the value of textarea elements, the line break 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).

11. Otherwise, append an entry to entry list with name and the value of the field element.

12. Otherwise, append an entry to entry list with name and the value of the field element.

13. If the element has a dirname attribute, and that attribute's value is not the empty string, then:

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 entry list with dirname and dir.

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.

6. Let form data be a new FormData object associated with entry list.

7. Fire an event named formdata at form using FormDataEvent, with the formData attribute initialized to form data and the bubbles attribute initialized to true.

8. Set form's constructing entry list to false.

9. Return a clone of entry list.

To append an entry to entry list, given name, value, and optional prevent line break normalization flag, run these steps:

1. For name, replace every occurrence of U+000D (CR) not followed by U+000A (LF), and every occurrence of U+000A (LF) not preceded by U+000D (CR), by a string consisting of a U+000D (CR) and U+000A (LF).
2. Set name to the result of converting name.
3. If value is not a File object, then:
   1. If the prevent line break normalization flag is unset, then replace every occurrence of U+000D (CR) not followed by U+000A (LF), and every occurrence of U+000A (LF) not preceded by U+000D (CR), by a string consisting of a U+000D (CR) and U+000A (LF).
   2. Set value to the result of converting value.
4. Create an entry with name and value, and append it to entry list.

4.10.21.5 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 ASCII whitespace.
   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.

4.10.21.6 URL-encoded form data

See URL for details on application/x-www-form-urlencoded. [URL]

4.10.21.7 Multipart form data

The multipart/form-data encoding algorithm, given an entry list and encoding, is as follows:

1. Return the byte sequence resulting from encoding the entry list using the rules described by RFC 7578, Returning Values from Forms: multipart/form-data, given the following conditions: [RFC7578]

   ◦ Each entry in entry list 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 entry list. Multiple entries with the same name must be treated as distinct fields.

   ◦ Field names, field values for non-file fields, and file names for file fields, in the generated multipart/form-data resource must be set to the result of encoding the corresponding entry's name or value with encoding, converted to a byte sequence.

   ◦ For field names and file names for file fields, the result of the encoding in the previous bullet point must be escaped by replacing any 0x0A (LF) bytes with the byte sequence `%0A`, 0x0D (CR) with `%0D` and 0x22 ("), with `%22`. The user agent must not perform any other escapes.

   ◦ The parts of the generated multipart/form-data resource that correspond to non-file fields must not have a `Content-Type` header specified.

   ◦ 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]

4.10.21.8 Plain text form data

The text/plain encoding algorithm, given an entry list, is as follows:

1. Let result be the empty string.

2. For each entry in entry list:

   1. If the entry's value is a File object, then set its value to the File object's name.

   2. Append the entry's name to result.

   3. Append a single U+003D EQUALS SIGN character (=) to result.

   4. Append the entry's value to result.

   5. Append a U+000D CARRIAGE RETURN (CR) U+000A LINE FEED (LF) character pair to result.

3. Return result.

Payloads using the text/plain format are intended to be human readable. They are not reliably interpretable by computer, as the
4.10.21.9 The `SubmitEvent` interface

```idl
[Exposed=Window]
interface SubmitEvent : Event {
    constructor(DOMString type, optional SubmitEventInit eventInitDict = {});

    readonly attribute HTMLElement? submitter;
}

dictionary SubmitEventInit : EventInit {
    HTMLElement? submitter = null;
}
```

For web developers (non-normative)

```plaintext
event.submitter
```

Returns the element representing the submit button that triggered the form submission, or null if the submission was not triggered by a button.

The `submitter` attribute must return the value it was initialized to.

4.10.21.10 The `FormDataEvent` interface

```idl
[Exposed=Window]
interface FormDataEvent : Event {
    constructor(DOMString type, FormDataEventInit eventInitDict);

    readonly attribute FormData formData;
}

dictionary FormDataEventInit : EventInit {
    required FormData formData;
}
```

For web developers (non-normative)

```plaintext
event.formData
```

Returns a `FormData` object representing names and values of elements associated to the target form. Operations on the `FormData` object will affect form data to be submitted.

The `formData` attribute must return the value it was initialized to. It represents a `FormData` object associated to the entry list that is constructed when the `form` is submitted.

4.10.22 Resetting a form

When a `form` element `form` is reset, run these steps:

1. Let `reset` be the result of firing an event named `reset` at `form`, with the `bubbles` and `cancelable` attributes initialized to true.

2. If `reset` is true, then invoke the `reset algorithm` of each `resettable element` whose `form owner` is `form`.

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

Accessibility considerations:
For authors.
For implementers.

DOM interface:

```javascript
[Exposed=Window]
interface HTMLDetailsElement : HTMLElement {
    [HTMLConstructor] constructor();

    [CFReactions] 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 the section on 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 represent 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 empty string. To honor a request for the information to be hidden, the user agent must remove the open attribute from the element.

**Note:**
This ability to request that additional information be shown or hidden may simply be the activation behavior of the appropriate summary element, in the case such an element exists. However, if no such element exists, user agents can still provide this ability through some other user interface affordance.
Whenever the `open` attribute is added to or removed from a `details` element, the user agent must queue an element task on the DOM manipulation task source given then `details` element 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 return.

   **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 an event named `toggle` at the `details` element.

The `open` IDL attribute must reflect the `open` content attribute.

**Example**

The following example shows the `details` element being used to hide technical details in a progress report.

```html
<section class="progress window">
  <h1>Copied "Really Achieving Your Childhood Dreams"</h1>
  <details>
    <summary>Copying...</summary>
    <progress max="375505392" value="97543282">25%</progress>
  </details>
  <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>
</section>
```

**Example**

The following shows how a `details` element can be used to hide some controls by default:

```html
<details>
  <summary><label for=fn>Name & Extension:</label></summary>
  <p><input type=text id=fn name=fn value="Pillar Magazine.pdf"></p>
  <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**

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 style sheet 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>
```
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:**
Phrasing content optionally intermixed with heading content.

**Tag omission in text/html:**
Neither tag is omissible.

**Content attributes:**
Global attributes

**Accessibility considerations:**
For authors.

For implementers.

**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.

A **summary** element is a **summary for its parent details** if the following algorithm returns true:

1. If this **summary** element has no parent, then return false.
2. Let parent be this **summary** element's parent.
3. If parent is not a **details** element, then return false.
4. If parent's first **summary** element child is not this **summary** element, then return false.
5. Return true.

The **activation behavior** of **summary** elements is to run the following steps:

1. If this **summary** element is not the **summary for its parent details**, then return.
2. Let parent be this **summary** element's parent.
3. If the **open** attribute is present on parent, then remove it. Otherwise, set parent's **open** attribute to the empty string.

**Note:**
This will then run the details notification task steps.

4.11.3 Commands

**4.11.3.1 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 with a non-null browsing context.
- Neither the element nor any of its ancestors has a hidden attribute specified.

User agents are encouraged to do this especially for commands that have Access Keys, as a way to advertise those keys to the user.

An `a` element with an `href` attribute defines a command.

The Label of the command is the element's descendant text content.

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 is to fire a click event at the element.

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 UA-dependent, locale-dependent value that the UA uses to label the button if the attribute is absent.

• Otherwise, if the element is a `labeled control`, then the `Label` is the `descendant text content` of the first `label` element in `tree order` whose `labeled control` is the element in question. (In JavaScript terms, this is 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.

**Note**

Even though the `value` attribute on `input` elements in the `Image Button` state is non-conformant, the attribute can still contribute to the `Label` determination, if it is present and the Image Button's `alt` attribute is missing.

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 is to fire a `click event` at the element.

### 4.11.3.5 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 `option` element's `descendant text content`, with ASCII whitespace stripped and collapsed.

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 `toggle` the `option` element. Otherwise, the `Action` is to `pick` the `option` element.

### 4.11.3.6 Using the `accesskey` attribute on a `legend` element to define a command

A `legend` element defines a command if all of the following are true:

- It has an `assigned access key`.
- It is a child of a `fieldset` element.
- Its parent has a descendant that defines a command that is neither a `label` element nor a `legend` element. This element, if it exists, is the `legend` element's `accesskey delegatee`.

The `Label` of the command is the element's `descendant text content`.

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 `legend element's accesskey delegate`. 609
In this example, the `legend` element specifies an `accesskey`, which, when activated, will delegate to the `input` element inside the `legend` element.

```html
<legend accesskey="p">
  <label> I want <input name="pizza" type="number" step="1" value="1" min="0"> pizza(s) with these toppings </label>
</legend>
<input name="pizza-cheese" type="checkbox" checked> Cheese
<input name="pizza-ham" type="checkbox" checked> Ham
<input name="pizza-pineapple" type="checkbox"> Pineapple

</fieldset>
```

### 4.11.3.7 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 descendant text content of the first label element in tree order whose labeled control is the element in question is the Label (in JavaScript terms, this is given by `element.labels[0].textContent`). Otherwise, the Label is the element's descendant text content.

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. Fire a click event at the element.

### 4.11.4 The `dialog` element

**Categories:** Flow 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
  - open — Whether the dialog box is showing

**Accessibility considerations:**

For authors.
For implementers.
The `dialog` element represents a part of an application that a user interacts with to perform a task, for example a dialog box, inspector, or window.

The `open` attribute is a boolean attribute. When specified, it indicates that the `dialog` element is active and that the user can interact with it.

A `dialog` element without an `open` attribute specified should not be shown to the user. This requirement may be implemented indirectly through the style layer. For example, user agents that support the suggested default rendering implement this requirement using the CSS rules described in the rendering section.

### Note

Removing the `open` attribute will usually hide the dialog. However, doing so has a number of strange additional consequences:

- The `close` event will not be fired.
- The `close` method, and any user-agent provided cancellation interface, will no longer be able to close the dialog.
- If the dialog was shown using its `showModal` method, the `Document` will still be blocked.

For these reasons, it is generally better to never remove the `open` attribute manually. Instead, use the `close` method to close the dialog, or the `hidden` attribute to hide it.

The `tabindex` attribute must not be specified on `dialog` elements.

### For web developers (non-normative)

#### `dialog . show()`

Displays the `dialog` element.

#### `dialog . showModal()`

Displays the `dialog` element and makes it the top-most modal dialog.

This method honors the `autofocus` attribute.

#### `dialog . close([ result ])`

Closes the `dialog` element.

The argument, if provided, provides a return value.

#### `dialog . returnValue` [ = result ]

Returns the `dialog`'s return value.

Can be set, to update the return value.

When the `show()` method is invoked, the user agent must run the following steps:

1. If the element already has a `open` attribute, then return.
2. Add an `open` attribute to the `dialog` element, whose value is the empty string.
3. Run the `dialog focusing steps` for the `dialog` element.

When the `showModal()` method is invoked, the user agent must run the following steps:
1. Let `subject` be the `dialog` element on which the method was invoked.
2. If `subject` already has an `open` attribute, then throw an "InvalidStateError" `DOMException`.
3. If `subject` is not `connected`, then throw an "InvalidStateError" `DOMException`.
4. Add an `open` attribute to `subject`, whose value is the empty string.
5. Set the `is modal` flag of `subject` to true.
6. Let `subject`'s `node document` be blocked by the modal dialog `subject`.

**Note**

This will cause the focused area of the document to become inert (unless that currently focused area is a shadow-including descendant of subject). In such cases, the focus fixup rule will kick in and reset the focused area of the document to the viewport for now. In a couple steps we will attempt to find a better candidate to focus.

7. If `subject`'s `node document`'s top layer does not already contain `subject`, then add `subject` to `subject`'s `node document`'s top layer.
8. Run the dialog focusing steps for `subject`.

The dialog focusing steps for a `dialog` element `subject` are as follows:

1. If `subject` is `inert`, return.
2. Let `control` be the first descendant element of `subject`, in tree order, that is not `inert` and has the `autofocus` attribute specified.
   
   If there isn't one, then let `control` be the first non-`inert` descendant element of `subject`, in tree order.
   
   If there isn't one of those either, then let `control` be `subject`.
3. Run the focusing steps for `control`.

**Note**

If `control` is not `focusable`, this will do nothing. For modal dialogs, this means that any earlier modifications to the focused area of the document will apply.

4. Let `topDocument` be the active document of `control`'s `node document`'s browsing context's top-level browsing context.
5. If `control`'s `node document`'s `origin` is not the same as the `origin` of `topDocument`, then return.
7. Set `topDocument`'s `autofocus processed flag` to true.

If at any time a `dialog` element is removed from a `Document`, then if that `dialog` is in that `Document`'s `top layer`, it must be removed from it.

When the `close()` method is invoked, the user agent must close the dialog that the method was invoked on. If the method was invoked with an argument, that argument must be used as the return value; otherwise, there is no return value.

When a `dialog` element `subject` is to be `closed`, optionally with a return value `result`, the user agent must run the following steps:

1. If `subject` does not have an `open` attribute, then return.
2. Remove `subject`'s `open` attribute.
3. Set the `is modal` flag of `subject` to false.
4. If the argument `result` was provided, then set the `returnValue` attribute to the value of `result`.
5. If `subject` is in its `Document`'s `top layer`, then remove it.
6. Queue an element task on the user interaction task source given the `subject` element to fire an event named `close` at `subject`. 


The `returnValue` IDL attribute, on getting, must return the last value to which it was set. On setting, it must be set to the new value. When the element is created, it must be set to the empty string.

**Canceling dialogs:** When `Document` is blocked by a modal dialog dialog, user agents may provide a user interface that, upon activation, queues an element task on the user interaction task source given the `dialog` element to run these steps:

1. Let `close` be the result of firing an event named `cancel` at `dialog`, with the `cancelable` attribute initialized to true.
2. If `close` is true and `dialog` has an `open` attribute, then close the `dialog` with no return value.

Note

An example of such a UI mechanism would be the user pressing the "Escape" key.

Each `dialog` element has an `is modal` flag. When a `dialog` element is created, this flag must be set to false.

The `open` IDL attribute must reflect the `open` content attribute.

**Example**

This dialog box has some small print. The `strong` element is used to draw the user's attention to the more important part.

```html
<dialog>
  <h1>Add to Wallet</h1>
  <p><strong>Label for amt</strong> How many gold coins do you want to add to your wallet? </p>
  <p><input id=amt name=amt type=number min=0 step=0.01 value=100></p>
  <p><small>You add coins at your own risk.</small></p>
  <label><input name=round type=checkbox> Only add perfectly round coins</label>
  <input type=button onclick="submit()" value="Add Coins"></p>
</dialog>
```

**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**

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**

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\(^{133}\).  
Script-supporting element\(^{134}\).  

Contexts in which this element can be used\(^{129}\):
Where metadata content\(^{132}\) is expected.  
Where phrasing content\(^{133}\) is expected.  
Where script-supporting elements\(^{134}\) are expected.

Content model\(^{129}\):
If there is no src\(^{615}\) attribute, depends on the value of the type\(^{614}\) attribute, but must match script content restrictions\(^{624}\).  
If there is a src\(^{615}\) attribute, the element must be either empty or contain only script documentation\(^{626}\) that also matches script content restrictions\(^{624}\).

Tag omission in text/html\(^{130}\):
Neither tag is omissible.

Content attributes\(^{130}\):
Global attributes\(^{137}\)
src\(^{615}\) — Address of the resource  
type\(^{614}\) — Type of script  
nomodule\(^{615}\) — Prevents execution in user agents that support module scripts\(^{911}\)  
async\(^{615}\) — Execute script when available, without blocking while fetching  
defer\(^{615}\) — Defer script execution  
crossorigin\(^{616}\) — How the element handles crossorigin requests  
integrity\(^{616}\) — Integrity metadata used in Subresource Integrity checks [SRI]\(^{1276}\)  
referrerpolicy\(^{616}\) — Referrer policy for fetches initiated by the element

Accessibility considerations\(^{130}\):
For authors.  
For implementers.

DOM interface\(^{130}\):

```
IDL [Exposed=Window]
interface HTMLScriptElement : HTMLElement {
    [HTMLConstructor] constructor();

    [CEReactions] attribute USVString src;
    [CEReactions] attribute DOMString type;
    [CEReactions] attribute boolean nomodule;
    [CEReactions] attribute boolean async;
    [CEReactions] attribute boolean defer;
    [CEReactions] attribute DOMString? crossorigin;
    [CEReactions] attribute DOMString text;
    [CEReactions] attribute DOMString integrity;
    [CEReactions] attribute DOMString referrerPolicy;

    // also has obsolete members
};
```

The script\(^{613}\) element allows authors to include dynamic script and data blocks in their documents. The element does not represent\(^{124}\) content for the user.

The type attribute allows customization of the type of script represented:

- Omitting the attribute, setting it to the empty string, or setting it to a JavaScript MIME type essence match, means that the script is a classic script\(^{911}\), to be interpreted according to the JavaScript Script top-level production. Classic scripts are affected by the async\(^{615}\) and defer\(^{615}\) attributes, but only when the src\(^{615}\) attribute is set. Authors should omit the type\(^{614}\) attribute instead of redundantly setting it.

- Setting the attribute to an ASCII case-insensitive match for the string "module" means that the script is a module script\(^{911}\) to be interpreted according to the JavaScript Module top-level production. Module scripts are not affected by the defer\(^{615}\) attribute, but are affected by the async\(^{615}\) attribute (regardless of the state of the src\(^{615}\) attribute).
• 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 string that is not a JavaScript MIME type essence match to denote data blocks.

Note

The requirement that data blocks must be denoted using a valid MIME type string 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, like how the "module" value denotes module scripts. By using a valid MIME type string now, you ensure that your data block will not ever be reinterpreted as a different script type, even in future user agents.

Classic scripts and module scripts can be embedded inline, or 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 or Module productions, for classic scripts and module scripts respectively.

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, async, nomodule, defer, crossorigin, integrity, and referrerpolicy attributes must not be specified.

The nomodule attribute is a boolean attribute that prevents a script from being executed in user agents that support module scripts. This allows selective execution of module scripts in modern user agents and classic scripts in older user agents, as shown below. The nomodule attribute must not be specified on module scripts (and will be ignored if it is).

The async and defer attributes are boolean attributes that indicate how the script should be evaluated. Classic scripts may specify defer or async, but must not specify either unless the src attribute is present. Module scripts may specify the async attribute, but must not specify the defer attribute.

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.

For module scripts, if the async attribute is present, then the module script and all its dependencies will be fetched in parallel to parsing, and the module script will be evaluated as soon as it is available (potentially before parsing completes). Otherwise, the module script and its dependencies will be fetched in parallel to parsing and evaluated when the page has finished parsing. (The defer attribute has no effect on module scripts.)

This is all summarized in the following schematic diagram:

```html
<script>
  Scripting:
  HTML Parser:
  ▼ parser ▶ fetch ▼ execution
</script>

<script defer>
  Scripting:
  HTML Parser:
  ▼ parser ▶ fetch ▼ execution
</script>

<script async>
  Scripting:
  HTML Parser:
  ▼ parser ▶ fetch ▼ execution
</script>

<script type="module">
  Scripting:
  HTML Parser:
  ▼ parser ▶ fetch ▼ execution
</script>

<script type="module" async>
  Scripting:
  HTML Parser:
  ▼ parser ▶ fetch ▼ execution
</script>
```

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 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. For module scripts, it controls the credentials mode used for cross-origin requests.

**Note**

Unlike classic scripts, module scripts require the use of the CORS protocol for cross-origin fetching.

The `integrity` attribute represents the integrity metadata for requests which this element is responsible for. The value is text. The `integrity` attribute must not be specified when the `src` attribute is not specified.

The `referrerPolicy` attribute is a referrer policy attribute. Its purpose is to set the referrer policy used when fetching the script, as well as any scripts imported from it.

Changing the `src`, `type`, `defer`, and `integrity` attributes dynamically has no direct effect; these attributes are only used at specific times described below.

The IDL attributes `src`, `type`, `defer`, and `integrity`, must each reflect the respective content attributes of the same name.

The `referrerPolicy` IDL attribute must reflect the `referrerPolicy` content attribute, limited to only known values.

The `crossOrigin` IDL attribute must reflect the `crossOrigin` content attribute, limited to only known values.

The `noModule` IDL attribute must reflect the `noModule` content attribute.

The `async` IDL attribute controls whether the element will execute asynchronously 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.

**For web developers (non-normative)**

```javascript
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 `text` attribute’s getter must return this `script` element’s child text content.

The `text` attribute’s setter must string replace all with the given value within this `script` element.

**Note**

When inserted using the `document.write()` method, `script` elements usually execute (typically blocking further script execution or HTML parsing). When inserted using the `innerHTML` and `outerHTML` attributes, they do not execute at all.

**Example**

In this example, two `script` elements are used. One embeds an external classic script, and the other includes some data as a data block.
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

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.

```html
<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="pricecalc" 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.pricecalc);</script>
</form>
</script>

Example

The following sample shows how a `script` element can be used to include an external `module script`. This module, and all its dependencies (expressed through JavaScript `import` statements in the source file), will be fetched. Once the entire resulting module graph has been imported, and the document has finished parsing, the contents of `app.mjs` will be evaluated.
Additionally, if code from another script element in the same Window imports the module from app.mjs (e.g. via import "./app.mjs";), then the same module script created by the former script element will be imported.

Example
This example shows how to include a module script for modern user agents, and a classic script for older agents:

```html
<script type="module" src="app.mjs"></script>
<script nomodule defer src="classic-app-bundle.js"></script>
```

In modern user agents that support module scripts, the script element with the nomodule attribute will be ignored, and the script element with a type of "module" will be fetched and evaluated (as a module script). Conversely, older user agents will ignore the script element with a type of "module", as that is an unknown script type for them — but they will have no problem fetching and evaluating the other script element (as a classic script), since they do not implement the nomodule attribute.

Example
The following sample shows how a script element can be used to write an inline module script that performs a number of substitutions on the document's text, in order to make for a more interesting reading experience (e.g. on a news site): [XKCD1288]

```javascript
import { walkAllTextNodeDescendants } from "./dom-utils.mjs";

const substitutions = new Map([  ["witnesses", "these dudes I know"],  ["allegedly", "kinda probably"],  ["new study", "Tumblr post"],  ["rebuild", "avenge"],  ["space", "spaaace"],  ["Google glass", "Virtual Boy"],  ["smartphone", "Pokédex"],  ["electric", "atomic"],  ["Senator", "Elf-Lord"],  ["car", "cat"],  ["election", "eating contest"],  ["Congressional leaders", "river spirits"],  ["homeland security", "Homestar Runner"],  ["could not be reached for comment", "is guilty and everyone knows it"]]);

function substitute(textNode) {
  for (const [before, after] of substitutions.entries()) {
    textNode.data = textNode.data.replace(new RegExp(`\b${before}\b`, "ig"), after);
  }
}

walkAllTextNodeDescendants(document.body, substitute);
```

Some notable features gained by using a module script include the ability to import functions from other JavaScript modules, strict mode by default, and how top-level declarations do not introduce new properties onto the global object. Also note that no matter where this script element appears in the document, it will not be evaluated until both document parsing has complete and its dependency (dom-utils.mjs) has been fetched and evaluated.
### 4.12.1.1 Processing model

A **script** element has several associated pieces of state.

A **script** element has a flag indicating whether or not it 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.

A **script** element has a **parser document**, which is either null or a **Document**. Initially, its value must be null. It is set by the HTML parser and the XML parser on **script** elements they insert, and affects the processing of those elements. **script** elements with non-null **parser document** are known as "parser-inserted".

A **script** element has a flag indicating whether the element will be "non-blocking". Initially, **script** elements must have this flag set. It is unset by the **HTML parser** and has a **XML parser** connected to it. The script is ready "parser-inserted" node document executing and a node or document fragment is prepared, based on the type attribute of the element at that time.

A **script** element has 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 **script's type** for a **script** element is either "classic" or "module". It is determined when the script is prepared, based on the **type** attribute of the element at that time.

A **script** element has a **preparation-time document**, which is a **Document** determined near the beginning of the prepare a script algorithm. It is used to prevent scripts that move between documents during preparation from executing.

A **script** element has 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.

The **script's script** for a **script** element is either null or a **script** resulting from preparing the element. This is set asynchronously after the classic script or module graph 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 "parser-inserted" experiences one of the events listed in the following list, the user agent must immediately prepare the **script** element:

- The **script** element becomes connected.
- The **script** element is connected and a node or document fragment is inserted into the **script** element, after any **script** elements inserted at that time.
- The **script** element is connected 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 return. The script is not executed.
2. Let parser document be the element's **parser document**.
3. Set the element's **parser document** to null.

**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.

4. If parser document is non-null 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
5. Let source text be the element's child text content.
6. If the element has no src attribute, and source text is the empty string, then return. The script is not executed.
7. If the element is not connected, then return. The script is not executed.
8. If either:
   ◦ the script element has a type attribute but it has a language attribute and that attribute's 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 leading and trailing ASCII whitespace stripped.
   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.

Determine the script's type as follows:
   ◦ If the script block's type string is a JavaScript MIME type essence match, the script's type is "classic".
   ◦ If the script block's type string is an ASCII case-insensitive match for the string "module", the script's type is "module".
   ◦ If neither of the above conditions are true, then return. No script is executed.
9. If parser document is non-null, then set the element's parser document back to parser document and set the element's "non-blocking" flag to false.
10. Set the element's "already started" flag.
11. Set the element's preparation-time document to its node document.
12. If parser document is non-null, and parser document is not equal to the element's preparation-time document, then return.
13. If scripting is disabled for the script element, then return. The script is not executed.

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][DOMPARSING][XSLTP][DOM].

This means specifying module on a module script has no effect; the algorithm continues onward.

14. If the script element has a nomodule content attribute and the script's type is "classic", then return. The script is not executed.

15. 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 source text, then return. The script is not executed.

16. If the script element has an event attribute and a for attribute, and the script's type is "classic", then:
   1. Let for be the value of the for attribute.
2. Let \textit{event} be the value of the \textit{event} attribute.
3. Strip leading and trailing ASCII whitespace from \textit{event} and \textit{for}.
4. If \textit{for} is not an ASCII case-insensitive match for the string "window", then return. The script is not executed.
5. If \textit{event} is not an ASCII case-insensitive match for either the string "onload" or the string "onload()", then return. The script is not executed.

17. If the \textit{script} element has a \textit{charset} attribute, then let \textit{encoding} be the result of getting an encoding from the value of the \textit{charset} attribute.

18. If the \textit{script} element does not have a \textit{charset} attribute, or if getting an encoding failed, let \textit{encoding} be the same as the encoding of the \textit{script} element's node document.

19. If the \textit{script} element has an \textit{integrity} attribute, then let \textit{integrity metadata} be that attribute's value.
Otherwise, let \textit{integrity metadata} be the empty string.

20. If the \textit{script} element has an \textit{integrity} attribute, then let \textit{integrity metadata} be that attribute's value.
Otherwise, let \textit{integrity metadata} be the empty string.

21. Let \textit{referrer policy} be the current state of the element's \textit{referrerpolicy} content attribute.

22. Let \textit{parser metadata} be "parser-inserted" if the \textit{script} element is "parser-inserted", and "not-parser-inserted" otherwise.

23. Let \textit{options} be a \textit{script fetch options} whose \textit{cryptographic nonce} is \textit{cryptographic nonce}, \textit{integrity metadata} is \textit{integrity metadata}, \textit{credentials mode} is \textit{module script credentials mode}, and \textit{referrer policy} is \textit{referrer policy}.

24. Let \textit{settings object} be the element's node document's relevant settings object.

25. If the element has a \textit{src} content attribute, then:

1. Let \textit{src} be the value of the element's \textit{src} attribute.
2. If \textit{src} is the empty string, queue a task to fire an event named \textit{error} at the element, and return.
3. Set the element's from an external file flag.
4. \textit{Parse} \textit{src} relative to the element's node document.
5. If the previous step failed, queue a task to fire an event named \textit{error} at the element, and return. Otherwise, let \textit{url} be the resulting URL record.
6. Switch on the \textit{script's type}:

→ "classic"

\textit{Fetch a classic script} given \textit{url}, \textit{settings object}, \textit{options}, classic script CORS setting, and \textit{encoding}.

→ "module"

\textit{Fetch an external module script graph} given \textit{url}, \textit{settings object}, and \textit{options}.

When the chosen algorithm asynchronously completes, set \textit{the script's script} to the result. At that time, \textit{the script is ready}.

For performance reasons, user agents may start fetching the classic script or module graph (as defined above) as soon as the \textit{src} attribute is set, instead, in the hope that the element will be inserted into the document (and that the \textit{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 \textit{src} attribute is dynamically changed, or the
If the element does not have a `src` content attribute, run these substeps:

1. Let `base URL` be the `script` element’s `node document`’s `document base URL`.
2. Switch on the `script's type`:
   - "classic"
     1. Let `script` be the result of creating a classic script using source text, settings object, base URL, and options.
     2. Set the `script's script` to `script`.
     3. The script is ready.
   - "module"
     1. Fetch an inline module script graph, given source text, base URL, settings object, and options. When this asynchronously completes, set the `script's script` to the result. At that time, the script is ready.

28. Then, follow the first of the following options that describes the situation:

- If the `script's type` is "classic", and the element has a `src` attribute, and the element has a `defer` attribute, and the element is "parser-inserted", and the element does not have an `async` attribute
  - If the `script's type` is "module", and the element is "parser-inserted", and the element does not have an `async` attribute
    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 `script is ready`, set the element’s "ready to be parser-executed" flag. The parser will handle executing the script.

- If the `script's type` is "classic", and the element has a `src` attribute, and the element is "parser-inserted", and the element does not have an `async` attribute
  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.

- If the `script's type` is "classic", and the element has a `src` attribute, and the element does not have an `async` attribute, and the element does not have the "non-blocking" flag set
  - If the `script's type` is "module", and the element does not have an `async` attribute, and the element does not have the "non-blocking" flag set
    Add the element to the end of the list of scripts that will execute in order as soon as possible associated with the element’s preparation-time document.
    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 return 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.
If the script's type is "classic", and the element has a src attribute

The element must be added to the set of scripts that will execute as soon as possible of the element's preparation-time document.

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.

If the element does not have a src attribute, and the element is "parser-inserted", and either the parser that created the script is an XML parser or it's an HTML parser whose script nesting level is not greater than one, and the element's parser document has a style sheet that is blocking scripts

The element is the pending parsing-blocking script of its parser document. (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).

To execute a script block given a script element scriptElement:

1. Let document be scriptElement's node document.
2. If scriptElement's preparation-time document is not equal to document, then return.
3. If the script's script is null for scriptElement, then fire an event named error at scriptElement, and return.
4. If scriptElement is from an external file, or the script's type for scriptElement is "module", then increment document's ignore-destructive-writes counter.
5. Switch on the script's type for scriptElement:

   "classic"

   1. Let oldCurrentScript be the value to which document's currentScript object was most recently set.
2. If scriptElement's root is not a shadow root, then set document's currentScript attribute to scriptElement. Otherwise, set it to null.

   Note

   This does not use the in a document tree check, as scriptElement could have been removed from the document prior to execution, and in that scenario currentScript still needs to point to it.
3. Run the classic script given by the script's script for scriptElement.
4. Set document's currentScript attribute to oldCurrentScript.

   "module"

1. Assert: document's currentScript attribute is null.
2. Run the module script given by the script's script for scriptElement.

6. Decrement the ignore-destructive-writes counter of document, if it was incremented in the earlier step.
7. If `scriptElement` is from an external file, then fire an event named `load` at `scriptElement`.

4.12.1.2 Scripting languages

User agents are not required to support JavaScript. This standard needs to be updated if a language other than JavaScript comes along and gets similar wide adoption by web browsers. Until such a time, implementing other languages is in conflict with this standard, given the processing model defined for the `script` element.

Servers should use `text/javascript` for JavaScript resources. Servers should not use other JavaScript MIME types for JavaScript resources, and must not use non-JavaScript MIME types.

For external JavaScript resources, MIME type parameters in `Content-Type` headers are generally ignored. (In some cases the `charset` parameter has an effect.) However, for the `script` element’s `type` attribute they are significant; it uses the JavaScript MIME type essence match concept.

For example, scripts with their `type` attribute set to "text/javascript; charset=utf-8" will not be evaluated, even though that is a valid JavaScript MIME type when parsed.

Furthermore, again for external JavaScript resources, special considerations apply around `Content-Type` header processing as detailed in the prepare a script algorithm and Fetch. [FETCH]

4.12.1.3 Restrictions for contents of `script` elements

The easiest and safest way to avoid the rather strange restrictions described in this section is to always escape an ASCII case-insensitive match for "<!-" 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 `script` element’s descendant text content must match the `script` production in the following ABNF, the character set for which is Unicode. [ABNF]
When a `<script>` element contains script documentation, there are further restrictions on the contents of the element, as described in the section below.

### Example

The following script illustrates this issue. Suppose you have a script that contains a string, as in:

```javascript
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:

```html
<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 (highlighted here) is not valid JavaScript:

```html
<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:

```html
<script>
// Note: '\s' is an escape sequence for 's'.
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:

```javascript
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:

```javascript
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 scalar value other than U+000A LINE FEED (LF)
not-star = %x0000-0029 / %x002B-10FFFF
        ; a scalar value other than U+002A ASTERISK (*)
not-slash = %x0000-002E / %x0030-10FFFF
        ; a scalar value 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**

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.

```html
<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>
```
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 implementers, 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 have its `parser document` set correctly, 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` object with a null `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 have their `parser document` set, 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 manually by creating the elements using `document.createElementNS()`. For instance, it needs to create `script` elements with null `parser document` 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 `Document`s 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

**Accessibility considerations**:

- For authors
- For implementers

**DOM interface**:

Uses `HTMLElement`.

The `noscript` element represents nothing if scripting is enabled, and 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 child text content 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.) [DOMPARSING]

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 XML 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 form submission, scripting, and so forth, even when scripting is enabled for the element.

Example
In the following example, a noscript element is used to provide fallback for a script.

```html
<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>
</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:

```html
<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 XML documents, since `<noscript>` is not allowed there.

### 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**: Nothing (for clarification, see example).

**Tag omission in text/html**: Neither tag is omissible.

**Content attributes**: Global attributes.
**Accessibility considerations**: For authors. For implementers.

**DOM interface**:

```javascript
[Exposed=Window]
interface HTMLTemplateElement : HTMLElement {
  [HTMLConstructor] constructor();
  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.

In a rendering, the `template` element represents nothing.

The `template contents` of a `template` element are not children of the element itself.

**Note** It is also possible, as a result of DOM manipulation, for a `template` element to contain Text nodes and element nodes; however, having any is a violation of the `template` element's content model, since its content model is defined as `nothing`.

**Example**

For example, consider the following document:

```html
<!doctype html>
<html lang="en">
<head>
  <title>Homework</title>
<body>
  <template id="template"><p>Smile!</p></template>
  <script>
    let num = 3;
    const fragment = document.getElementById('template').content.cloneNode(true);
    while (num-- > 1) {
      fragment.firstChild.before(fragment.firstChild.cloneNode(true));
      fragment.firstChild.textContent += fragment.lastChild.textContent;
    }
    document.body.appendChild(fragment);
  </script>
</body>
</html>
```

The `p` element in the `template` is not a child of the `template` in the DOM; it is a child of the DocumentFragment returned by the `template` element's `content` IDL attribute.

If the script were to call `appendChild()` on the `template` element, that would add a child to the `template` element (as for any other element); however, doing so is a violation of the `template` element's content model.

**For web developers (non-normative)**

`template . content` returns the `template contents` (a DocumentFragment).

Each `template` element has an associated DocumentFragment object that is its `template contents`. The `template contents` have no conformance requirements. 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`. The appropriate template contents owner document for `template`.
2. Create a DocumentFragment object whose `node document` is `doc` and `host` is the `template` element.
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, then:
   1. Let new `doc` be a new `Document` (whose browsing context is null). 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.

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.

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, return.

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

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.

```html
<!DOCTYPE html>
<html lang="en">
<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>
</tr>
</thead>
</table>
</html>
```
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 implementers, 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.

4.12.3.1 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 implementers, 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 slot 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**
  - `name` — Name of shadow tree slot

Accessibility considerations:

- **For authors**
- **For implementers**

DOM interface:

```idl
[Exposed=Window]
interface HTMLSlotElement : HTMLElement {
    [HTMLConstructor] constructor();

    [CEReactions] attribute DOMString name;
    sequence<Node> assignedNodes(optional AssignedNodesOptions options = {});
    sequence<Element> assignedElements(optional AssignedNodesOptions options = {});
}

dictionary AssignedNodesOptions {
    boolean flatten = false;
}
```

The `slot` element defines a `slot`. It is typically used in a shadow tree. A `slot` element represents its assigned nodes, if any, and its contents otherwise.

The `name` content attribute may contain any string value. It represents a `slot`'s `name`.

**Note**

The `name` attribute is used to assign slots to other elements: a `slot` element with a `name` attribute creates a named `slot` to which any element is assigned if that element has a `slot` attribute whose value matches that `name` attribute's value, and the `slot` element is a child of the shadow tree whose root's `host` has that corresponding `slot` attribute value.

**For web developers (non-normative)**

- `slot . name` Can be used to get and set slot's `name`.
- `slot . assignedNodes()` Returns slot's assigned nodes.
- `slot . assignedNodes({ flatten: true })` Returns slot's assigned nodes, if any, and slot's children otherwise, and does the same for any `slot` elements encountered therein, recursively, until there are no `slot` elements left.
- `slot . assignedElements()` Returns slot's assigned nodes, limited to elements.
- `slot . assignedElements({ flatten: true })` Returns the same as `assignedNodes({ flatten: true })`, limited to elements.

The `name` IDL attribute must reflect the content attribute of the same name.

The `assignedNodes(options)` method, when invoked, must run these steps:

1. If the value of `options`'s `flatten` member is false, then return this element's assigned nodes.
2. Return the result of finding flattened slottables with this element.

The `assignedElements(options)` method, when invoked, must run these steps:

1. If the value of `options`'s `flatten` member is false, then return this element's assigned nodes, filtered to contain only `Element` nodes.
2. Return the result of finding flattened slottables with this element, filtered to contain only Element nodes.

4.12.5 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, but with no interactive content descendants except for a, img elements with usemap attributes, button, input elements whose type attribute are in the Checkbox or Radio Button states, input elements that are buttons, and select elements with a multiple attribute or a display size greater than 1.

Tag omission in text/html:
Neither tag is omissible.

Content attributes:
- Global attributes
  - width — Horizontal dimension
  - height — Vertical dimension

Accessibility considerations:
- For authors.
- For implementers.

DOM interface:

```idl
typedef (CanvasRenderingContext2D or Image_bitmapRenderingContext or WebGLRenderingContext or WebGL2RenderingContext) RenderingContext;

[Exposed=Window]
interface HTMLCanvasElement : HTMLElement {
  [HTMLConstructor] constructor();

  [CFReactions] attribute unsigned long width;
  [CFReactions] attribute unsigned long height;

  RenderingContext? getContext(DOMString contextId, optional any options = null);

  USVString toDataURL(optional DOMString type = "image/png", optional any quality);
  undefined toBlob(BlobCallback _callback, optional DOMString type = "image/png", optional any quality);
  OffscreenCanvas transferControlToOffscreen();
};

callback BlobCallback = undefined (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 shadow trees.

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 can 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, then the `canvas` element represents, consisting of a dynamically created image, the element's bitmap.

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, 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, 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.)

An element whose nearest `canvas` element ancestor is being rendered and represents, 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.

When setting the value of the `width` or `height` attribute, if the context mode of the `canvas` element is set to placeholder, the user agent must throw an `InvalidStateError` DOMException and leave the attribute's value unchanged.

The intrinsic dimensions of the `canvas` element when it represents, 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.

A `canvas` element can be sized arbitrarily by a style sheet, its bitmap is then subject to the 'object-fit' CSS property.

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 sections on the `CanvasRenderingContext2D` and `ImageBitmapRenderingContext` objects 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` 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` has a context mode, which is initially none but can be changed to either `placeholder`, `2d`, `bitmaprenderer`, `webgl`, or `webgl2` by algorithms defined in this specification.

When its context mode is none, a `canvas` element has no rendering context, and its bitmap must be 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 its context mode is `placeholder`, a `canvas` element has no rendering context. It serves as a placeholder for an `OffscreenCanvas` object, and the content of the `canvas` element is updated by calling the `commit()` method of the `OffscreenCanvas` object's rendering context.

When a `canvas` element represents, 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, then the user agent must perform the action from the row of the following table that corresponds to the canvas element's context mode.

<table>
<thead>
<tr>
<th>Context Mode</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>2d</td>
<td>Follow the steps to set bitmap dimensions to the numeric values of the width and height content attributes.</td>
</tr>
<tr>
<td>webgl or webgl2</td>
<td>Follow the behavior defined in the WebGL specifications. [WEBGL]1277</td>
</tr>
<tr>
<td>bitmaprenderer</td>
<td>If the context's bitmap mode is set to blank, run the steps to set an ImageBitmapRenderingContext's output bitmap, passing the canvas element's rendering context.</td>
</tr>
<tr>
<td>placeholder</td>
<td>Do nothing.</td>
</tr>
<tr>
<td>none</td>
<td>Do nothing.</td>
</tr>
</tbody>
</table>

The width and height IDL attributes must reflect the respective content attributes of the same name, with the same defaults.

For web developers (non-normative)

```javascript
context = canvas.getContext(contextId [, options])
```

Returns an object that exposes an API for drawing on the canvas. contextId specifies the desired API: "2d", "bitmaprenderer", "webgl", or "webgl2". options is handled by that API.

This specification defines the "2d" and "bitmaprenderer" contexts below. The WebGL specifications define the "webgl" and "webgl2" contexts. [WEBGL]1277

Returns null if contextId is not supported, or if the canvas has already been initialized with another context type (e.g., trying to get a "2d" context after getting a "webgl" context).

The `getContext(contextId, options)` method of the canvas element, when invoked, must run these steps:

1. If options is not an object, then set options to null.
2. Set options to the result of converting options to a JavaScript value.
3. Run the steps in the cell of the following table whose column header matches this canvas element’s canvas context mode and whose row header matches contextId:

<table>
<thead>
<tr>
<th>Context Mode</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>Return null.</td>
</tr>
<tr>
<td>2d</td>
<td>Return the same object as was returned the last time the method was invoked with this same first argument.</td>
</tr>
<tr>
<td>bitmaprenderer</td>
<td>Return null.</td>
</tr>
<tr>
<td>webgl or webgl2</td>
<td>Return null.</td>
</tr>
<tr>
<td>[bitmaprenderer]</td>
<td>Return null.</td>
</tr>
<tr>
<td>[webgl] or [webgl2]</td>
<td>Throw an &quot;InvalidStateError&quot; DOMException.</td>
</tr>
</tbody>
</table>

For web developers (non-normative)

```javascript
context = canvas.getContext("2d")
```

Returns a CanvasRenderingContext2D object.

The canvas element’s 2D context has an API for drawing on the canvas. It exposes an object that exposes an API for drawing on the canvas. contextId specifies the desired API: "2d", "bitmaprenderer", "webgl", or "webgl2". options is handled by that API.

This specification defines the "2d" and "bitmaprenderer" contexts below. The WebGL specifications define the "webgl" and "webgl2" contexts. [WEBGL]1277

Returns null if contextId is not supported, or if the canvas has already been initialized with another context type (e.g., trying to get a "2d" context after getting a "webgl" context).

The `getContext("2d")` method of the canvas element, when invoked, must run these steps:

1. If options is not an object, then set options to null.
2. Set options to the result of converting options to a JavaScript value.
3. Run the steps in the cell of the following table whose column header matches this canvas element’s canvas context mode and whose row header matches contextId:

<table>
<thead>
<tr>
<th>Context Mode</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>Return null.</td>
</tr>
<tr>
<td>2d</td>
<td>Return the same object as was returned the last time the method was invoked with this same first argument.</td>
</tr>
<tr>
<td>bitmaprenderer</td>
<td>Return null.</td>
</tr>
<tr>
<td>webgl or webgl2</td>
<td>Return null.</td>
</tr>
<tr>
<td>[bitmaprenderer]</td>
<td>Return null.</td>
</tr>
<tr>
<td>[webgl] or [webgl2]</td>
<td>Throw an &quot;InvalidStateError&quot; DOMException.</td>
</tr>
</tbody>
</table>

For web developers (non-normative)

```javascript
context = canvas.getContext("bitmaprenderer")
```

Returns an object that exposes an API for drawing on the canvas. contextId specifies the desired API: "2d", "bitmaprenderer", "webgl", or "webgl2". options is handled by that API.

This specification defines the "2d" and "bitmaprenderer" contexts below. The WebGL specifications define the "webgl" and "webgl2" contexts. [WEBGL]1277

Returns null if contextId is not supported, or if the canvas has already been initialized with another context type (e.g., trying to get a "2d" context after getting a "webgl" context).

The `getContext("bitmaprenderer")` method of the canvas element, when invoked, must run these steps:

1. If options is not an object, then set options to null.
2. Set options to the result of converting options to a JavaScript value.
3. Run the steps in the cell of the following table whose column header matches this canvas element’s canvas context mode and whose row header matches contextId:

<table>
<thead>
<tr>
<th>Context Mode</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>Return null.</td>
</tr>
<tr>
<td>2d</td>
<td>Return the same object as was returned the last time the method was invoked with this same first argument.</td>
</tr>
<tr>
<td>bitmaprenderer</td>
<td>Return null.</td>
</tr>
<tr>
<td>webgl or webgl2</td>
<td>Return null.</td>
</tr>
<tr>
<td>[bitmaprenderer]</td>
<td>Return null.</td>
</tr>
<tr>
<td>[webgl] or [webgl2]</td>
<td>Throw an &quot;InvalidStateError&quot; DOMException.</td>
</tr>
<tr>
<td><strong>&quot;webgl2&quot;</strong>, if the user agent supports the WebGL feature in its current configuration</td>
<td>none</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>specifications' Context Creation sections to obtain a WebGLRenderingContext, WebGL2RenderingContext, or null; if the returned value is null, then return null; otherwise, set this canvas element's context mode to webgl or webgl2, and return the WebGLRenderingContext or WebGL2RenderingContext object. [WEBGL]</td>
<td>Return null.</td>
</tr>
</tbody>
</table>

* For example, the "webgl1" or "webgl2" value in the case of a user agent having exhausted the graphics hardware's abilities and having no software fallback implementation.

---

For web developers (non-normative)

```javascript
url = canvas . toDataURL() ( [ type [, quality ] ] )
```

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 second argument applies if the type is an image format that supports variable quality (such as "image/jpeg"), and is a number in the range 0.0 to 1.0 inclusive indicating the desired quality level for the resulting image.

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:,").

```javascript
canvas . toBlob(callback [, type [, quality ] ] )
```

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 third argument applies if the type is an image format that supports variable quality (such as "image/jpeg"), and is a number in the range 0.0 to 1.0 inclusive indicating the desired quality level for the resulting image.

```javascript
canvas . transferControlToOffscreen()
```

Returns a newly created OffscreenCanvas object that uses the canvas element as a placeholder. Once the canvas element has become a placeholder for an OffscreenCanvas object, its intrinsic size can no longer be changed, and it cannot have a rendering context. The content of the placeholder canvas is updated by calling the commit method of the OffscreenCanvas object's rendering context.

The `toDataURL(type, quality)` method, when invoked, must run these steps:

1. If this `canvas` element's bitmap's `origin-clean` flag is set to false, then throw a "SecurityError" DOMException.
2. If this `canvas` element's bitmap has no pixels (i.e. either its horizontal dimension or its vertical dimension is zero) then return the string "data:;". (This is the shortest data: URL; it represents the empty string in a text/plain resource.)
3. Let file be a serialization of this canvas element's bitmap as a file, passing type and quality if given.
4. If file is null then return "data:;".
5. Return a data: URL representing file. [RFC2397]

The `toBlob(callback, type, quality)` method, when invoked, must run these steps:

1. If this `canvas` element's bitmap's `origin-clean` flag is set to false, then throw a "SecurityError" DOMException.
2. Let result be null.
3. If this `canvas` element's bitmap has pixels (i.e., neither its horizontal dimension nor its vertical dimension is zero), then set result to a copy of this `canvas` element's bitmap.

---

637
4. Run these steps in parallel:

1. If `result` is non-null, then set `result` to a serialization of `result` as a file with type and quality if given.

2. Queue an element task on the canvas blob serialization task source given the canvas element to run these steps:
   1. If `result` is non-null, then set `result` to a new Blob object, created in the relevant Realm of this canvas element, representing `result`. [FILEAPI]
   2. Invoke callback with `result`.

The `transferControlToOffscreen()` method, when invoked, must run these steps:

1. If this canvas element's context mode is not set to `none`, throw an "InvalidStateError" DOMException.

2. Let `offscreenCanvas` be a new OffscreenCanvas object with its width and height equal to the values of the width and height content attributes of this canvas element.

3. Set the placeholder canvas element of `offscreenCanvas` to be a weak reference to this canvas element.

4. Set this canvas element's context mode to `placeholder`.

5. Return `offscreenCanvas`.

4.12.5.1 The 2D rendering context

```idl
typedef (HTMLImageElement or SVGImageElement) HTMLOrSVGImageElement;

typedef (HTMLOrSVGImageElement or HTMLVideoElement or HTMLCanvasElement or ImageBitmap or OffscreenCanvas) CanvasImageSource;

enum CanvasFillRule { "nonzero", "evenodd" ];

dictionary CanvasRenderingContext2DSettings { 
  boolean alpha = true;
  boolean desynchronized = false;
};

enum ImageSmoothingQuality { "low", "medium", "high" ];

[Exposed=Window]
interface CanvasRenderingContext2D { 
  // back-reference to the canvas
  readonly attribute HTMLCanvasElement canvas;

  CanvasRenderingContext2DSettings getContextAttributes();
};

CanvasRenderingContext2D includes CanvasState;
CanvasRenderingContext2D includes CanvasTransform;
CanvasRenderingContext2D includes CanvasCompositing;
CanvasRenderingContext2D includes CanvasImageSmoothing;
CanvasRenderingContext2D includes CanvasFillStrokeStyles;
CanvasRenderingContext2D includes CanvasShadowStyles;
CanvasRenderingContext2D includes CanvasFilters;
CanvasRenderingContext2D includes CanvasRect;
CanvasRenderingContext2D includes CanvasDrawPath;
CanvasRenderingContext2D includes CanvasUserInterface;
```
interface mixin CanvasState {
    // state
    undefined save(); // push state on state stack
    undefined restore(); // pop state stack and restore state
};

interface mixin CanvasTransform {
    // transformations (default transform is the identity matrix)
    undefined scale(unrestricted double x, unrestricted double y);
    undefined rotate(unrestricted double angle);
    undefined translate(unrestricted double x, unrestricted double y);
    undefined transform(unrestricted double a, unrestricted double b,
                        unrestricted double c, unrestricted double d,
                        unrestricted double e, unrestricted double f);

    [NewObject] DOMMatrix getTransform();
    undefined setTransform(unrestricted double a, unrestricted double b,
                            unrestricted double c, unrestricted double d,
                            unrestricted double e, unrestricted double f);
    undefined setTransform(optional DOMMatrix2DInit transform = {});
    undefined resetTransform();
};

interface mixin CanvasCompositing {
    // compositing
    attribute unrestricted double globalAlpha; // (default 1.0)
    attribute DOMString globalCompositeOperation; // (default source-over)
};

interface mixin CanvasImageSmoothing {
    // image smoothing
    attribute boolean imageSmoothingEnabled; // (default true)
    attribute ImageSmoothingQuality imageSmoothingQuality; // (default low)
};

interface mixin CanvasFillStrokeStyles {
    // colors and styles (see also the CanvasPathDrawingStyles and
    // CanvasTextDrawingStyles interfaces)
    attribute (DOMString or CanvasGradient or CanvasPattern) strokeStyle; // (default black)
    attribute (DOMString or CanvasGradient or CanvasPattern) fillStyle; // (default black)
    CanvasGradient createLinearGradient(double x0, double y0, double x1, double y1);
    CanvasGradient createRadialGradient(double x0, double y0, double r0, double x1, double y1, double r1);
    CanvasPattern createPattern(CanvasImageSource image, [LegacyNullToEmptyString] DOMString repetition);
};

interface mixin CanvasShadowStyles {
    // shadows
    attribute unrestricted double shadowOffsetX; // (default 0)
    attribute unrestricted double shadowOffsetY; // (default 0)
    attribute unrestricted double shadowBlur; // (default 0)
    attribute DOMString shadowColor; // (default transparent black)
};

interface mixin CanvasFilters {
interface mixin CanvasRect {
    // rects
    undefined clearRect(unrestricted double x, unrestricted double y, unrestricted double w, unrestricted double h);
    undefined fillRect(unrestricted double x, unrestricted double y, unrestricted double w, unrestricted double h);
    undefined strokeRect(unrestricted double x, unrestricted double y, unrestricted double w, unrestricted double h);
}

interface mixin CanvasDrawPath {
    // path API (see also CanvasPath)
    undefined beginPath();
    undefined fill(optional CanvasFillRule fillRule = "nonzero");
    undefined fill(Path2D path, optional CanvasFillRule fillRule = "nonzero");
    undefined stroke();
    undefined stroke(Path2D path);
    undefined clip(optional CanvasFillRule fillRule = "nonzero");
    undefined clip(Path2D path, optional CanvasFillRule fillRule = "nonzero");
    boolean isPointInPath(unrestricted double x, unrestricted double y, optional CanvasFillRule fillRule = "nonzero");
    boolean isPointInPath(Path2D path, unrestricted double x, unrestricted double y, optional CanvasFillRule fillRule = "nonzero");
    boolean isPointInStroke(unrestricted double x, unrestricted double y);
    boolean isPointInStroke(Path2D path, unrestricted double x, unrestricted double y);
}

interface mixin CanvasUserInterface {
    undefined drawFocusIfNeeded(Element element);
    undefined drawFocusIfNeeded(Path2D path, Element element);
    undefined scrollPathIntoView();
    undefined scrollPathIntoView(Path2D path);
}

interface mixin CanvasText {
    // text (see also the CanvasPathDrawingStyles and CanvasTextDrawingStyles interfaces)
    undefined fillText(DOMString text, unrestricted double x, unrestricted double y, optional unrestricted double maxWidth);
    undefined strokeText(DOMString text, unrestricted double x, unrestricted double y, optional unrestricted double maxWidth);
    TextMetrics measureText(DOMString text);
}

interface mixin CanvasDrawImage {
    // drawing images
    undefined drawImage(CanvasImageSource image, unrestricted double dx, unrestricted double dy);
    undefined drawImage(CanvasImageSource image, unrestricted double dx, unrestricted double dy, unrestricted double dw, unrestricted double dh);
    undefined drawImage(CanvasImageSource image, unrestricted double sx, unrestricted double sy, unrestricted double sw, unrestricted double sh, unrestricted double dx, unrestricted double dy, unrestricted double dw, unrestricted double dh);
}

interface mixin CanvasImageData {
    // pixel manipulation
    ImageData createImageData([EnforceRange] long sw, [EnforceRange] long sh);
    ImageData createImageData(ImageData imagedata);
undefined putImageData(Data imagedata, [EnforceRange] long dx, [EnforceRange] long dy);
undefined putImageData(Data imagedata, [EnforceRange] long dx, [EnforceRange] long dy,
[EnforceRange] long dirtyHeight);
};
enum CanvasLineCap { "butt", "round", "square" }
enum CanvasLineJoin { "round", "bevel", "miter" }
enum CanvasTextAlign { "start", "end", "left", "right", "center" }
enum CanvasTextBaseline { "top", "hanging", "middle", "alphabetic", "ideographic", "bottom" }
enum CanvasDirection { "ltr", "rtl", "inherit" }

interface mixin CanvasPathDrawingStyles {
   // line caps/joins
   attribute unrestricted double lineWidth; // (default 1)
   attribute CanvasLineCap lineCap; // (default "butt")
   attribute CanvasLineJoin lineJoin; // (default "miter")
   attribute unrestricted double miterLimit; // (default 10)

   // dashed lines
   undefined setLineDash(sequence<unrestricted double> segments); // default empty
   sequence<unrestricted double> getLineDash();
   attribute unrestricted double lineDashOffset;
};

interface mixin CanvasTextDrawingStyles {
   // text
   attribute DOMString font; // (default 10px sans-serif)
   attribute CanvasTextAlign textAlign; // (default: "start")
   attribute CanvasTextBaseline textBaseline; // (default: "alphabetic")
   attribute CanvasDirection direction; // (default: "inherit")
};

interface mixin CanvasPath {
   // shared path API methods
   undefined closePath();
   undefined moveTo(unrestricted double x, unrestricted double y);
   undefined lineTo(unrestricted double x, unrestricted double y);
   undefined quadraticCurveTo(unrestricted double cpx, unrestricted double cpy, unrestricted double x,
unrestricted double y);
   undefined bezierCurveTo(unrestricted double cpx1, unrestricted double cpy1, unrestricted double cpx2,
unrestricted double cpy2, unrestricted double x, unrestricted double y);
   undefined arcTo(unrestricted double x1, unrestricted double y1, unrestricted double x2, unrestricted
double y2, unrestricted double radius);
   undefined rect(unrestricted double x, unrestricted double y, unrestricted double w, unrestricted
double h);
   undefined arc(unrestricted double x, unrestricted double y, unrestricted double radius, unrestricted
double startAngle, unrestricted double endAngle, optional boolean counterclockwise = false);
   undefined ellipse(unrestricted double x, unrestricted double y, unrestricted double radiusX,
unrestricted double radiusY, unrestricted double rotation, unrestricted double startAngle, unrestricted
double endAngle, optional boolean counterclockwise = false);
};

[Exposed=(Window,Worker)]
interface CanvasGradient {
   // opaque object
   undefined addColorStop(double offset, DOMString color);
};
interface CanvasPattern {
    // opaque object
    undefined setTransform(optional DOMMatrix2DInit transform = {});
};

interface TextMetrics {
    // x-direction
    readonly attribute double width; // advance width
    readonly attribute double actualBoundingBoxLeft;
    readonly attribute double actualBoundingBoxRight;

    // y-direction
    readonly attribute double fontSizeAscent;
    readonly attribute double fontSizeDescent;
    readonly attribute double actualBoundingBoxAscent;
    readonly attribute double actualBoundingBoxDescent;
    readonly attribute double emHeightAscent;
    readonly attribute double emHeightDescent;
    readonly attribute double hangingBaseline;
    readonly attribute double alphabeticBaseline;
    readonly attribute double ideographicBaseline;
};

interface ImageData {
    constructor(unsigned long sw, unsigned long sh);
    constructor(Uint8ClampedArray data, unsigned long sw, optional unsigned long sh);

    readonly attribute unsigned long width;
    readonly attribute unsigned long height;
    readonly attribute Uint8ClampedArray data;
};

interface Path2D {
    constructor(optional (Path2D or DOMString) path);

    undefined addPath(Path2D path, optional DOMMatrix2DInit transform = {});
};

Path2D includes CanvasPath;

To maintain compatibility with existing web content, user agents need to enumerate methods defined in CanvasUserInterface immediately after the stroke() method on CanvasRenderingContext2D objects.

For web developers (non-normative)

context = canvas.getContext('2d', { [ alpha: true ], [ desynchronized: false ] });

Returns a CanvasRenderingContext2D object that is permanently bound to a particular canvas element.
If the alpha member is false, then the context is forced to always be opaque.
If the desynchronized member is true, then the context might be desynchronized.

canvas

Returns the canvas element.

attributes = canvas.getContextAttributes()

Returns an object whose:
A `CanvasRenderingContext2D` object has an **output bitmap** that is initialized when the object is created.

The **output bitmap** has an **origin-clean** flag, which can be set to true or false. Initially, when one of these bitmaps is created, its **origin-clean** flag must be set to true.

The `CanvasRenderingContext2D` object also has an **alpha** boolean. When a `CanvasRenderingContext2D` object's **alpha** is false, then its alpha channel must be fixed to 1.0 (fully opaque) for all pixels, and attempts to change the alpha component of any pixel must be silently ignored.

The `CanvasRenderingContext2D` object also has a **desynchronized** boolean. When a `CanvasRenderingContext2D` object's **desynchronized** is true, then the user agent may optimize the rendering of the canvas to reduce the latency, as measured from input events to rasterization, by desynchronizing the canvas paint cycle from the event loop, bypassing the ordinary user agent rendering algorithm, or both. Insofar as this mode involves bypassing the usual paint mechanisms, rasterization, or both, it might introduce visible tearing artifacts.

Note

*The bitmap of such a context starts off as opaque black instead of transparent black. `clearRect()` always results in opaque black pixels, every fourth byte from `getImageData()` is always 255, the `putImageData()` method effectively ignores every fourth byte in its input, and so on. However, the alpha component of styles and images drawn onto the canvas are still honoured up to the point where they would impact the output bitmap's alpha channel; for instance, drawing a 50% transparent white square on a freshly created output bitmap with its alpha set to false will result in a fully-opaque gray square.*

The `CanvasRenderingContext2D` object also has a **desynchronized** boolean. When a `CanvasRenderingContext2D` object's **desynchronized** is true, then the user agent may optimize the rendering of the canvas to reduce the latency, as measured from input events to rasterization, by desynchronizing the canvas paint cycle from the event loop, bypassing the ordinary user agent rendering algorithm, or both. Insofar as this mode involves bypassing the usual paint mechanisms, rasterization, or both, it might introduce visible tearing artifacts.

Note

*The user agent usually renders on a buffer which is not being displayed, quickly swapping it and the one being scanned out for presentation; the former buffer is called back buffer and the latter front buffer. A popular technique for reducing latency is called front buffer rendering, also known as single buffer rendering, where rendering happens in parallel and racily with the scanning out process. This technique reduces the latency at the price of potentially introducing tearing artifacts and can be used to implement in total or part of the desynchronized boolean.* [MULTIPLEBUFFERING]

Note

*The desynchronized boolean can be useful when implementing certain kinds of applications, such as drawing applications, where the latency between input and rasterization is critical.*

The `getContextAttributes()` method, when invoked, must return a `CanvasRenderingContext2DSettings` dictionary containing the following members:

- **alpha**, set to this context's **alpha**.
- **desynchronized**, set to this context's **desynchronized**.

The `CanvasRenderingContext2D` 2D rendering context represents a flat linear Cartesian surface whose origin (0,0) is at the top left corner, with the coordinate space having x values increasing when going right, and y values increasing when going down. The x-coordinate of the right-most edge is equal to the width of the rendering context's **output bitmap** in CSS pixels; similarly, the y-coordinate of the bottom-most edge is equal to the height of the rendering context's **output bitmap** in CSS pixels.

The size of the coordinate space does not necessarily represent the size of the actual bitmaps that the user agent will use internally or during rendering. On high-definition displays, for instance, the user agent may internally use bitmaps with four device pixels per unit in the coordinate space, so that the rendering remains at high quality throughout. Anti-aliasing can similarly be implemented using oversampling with bitmaps of a higher resolution than the final image on the display.

Example

Using **CSS pixels** to describe the size of a rendering context's **output bitmap** does not mean that when rendered the canvas will
cover an equivalent area in CSS pixels. CSS pixels are reused for ease of integration with CSS features, such as text layout.

In other words, the canvas element below's rendering context has a 200x200 output bitmap (which internally uses CSS pixels as a unit for ease of integration with CSS) and is rendered as 100x100 CSS pixels:

```html
<canvas width=200 height=200 style=width:100px;height:100px>
</canvas>
```

The **2D context creation algorithm**, which is passed a target (a canvas element) and options, consists of running these steps:

1. Let settings be the result of converting options to the dictionary type CanvasRenderingContext2DSettings. (This can throw an exception.).
2. Let context be a new CanvasRenderingContext2D object.
3. Initialize context's canvas attribute to point to target.
4. Set context's output bitmap to the same bitmap as target's bitmap (so that they are shared).
5. Set bitmap dimensions to the numeric values of target's width and height content attributes.
6. Set context's alpha to settings's alpha.
7. Set context's desynchronized to settings's desynchronized.
8. Return context.

When the user agent is to set bitmap dimensions to width and height, it must run these steps:

1. Reset the rendering context to its default state.
2. Resize the output bitmap to the new width and height and clear it to transparent black.
3. Let canvas be the canvas element to which the rendering context's canvas attribute was initialized.
4. If the numeric value of canvas's width content attribute differs from width, then set canvas's width content attribute to the shortest possible string representing width as a valid non-negative integer.
5. If the numeric value of canvas's height content attribute differs from height, then set canvas's height content attribute to the shortest possible string representing height as a valid non-negative integer.

The canvas attribute must return the value it was initialized to when the object was created.

The CanvasFillRule enumeration is used to select the fill rule algorithm by which to determine if a point is inside or outside a path.

The "nonzero" value indicates the nonzero winding rule, wherein a point is considered to be outside a shape if the number of times a half-infinite straight line drawn from that point crosses the shape's path going in one direction is equal to the number of times it crosses the path going in the other direction.
The "evenodd" value indicates the even-odd rule, wherein a point is considered to be outside a shape if the number of times a half-infinite straight line drawn from that point crosses the shape's path is even.

If a point is not outside a shape, it is inside the shape.

The `ImageSmoothingQuality` enumeration is used to express a preference for the interpolation quality to use when smoothing images.

The "low" value indicates a preference for a low level of image interpolation quality. Low-quality image interpolation may be more computationally efficient than higher settings.

The "medium" value indicates a preference for a medium level of image interpolation quality.

The "high" value indicates a preference for a high level of image interpolation quality. High-quality image interpolation may be more computationally expensive than lower settings.

**Note**

Bilinear scaling is an example of a relatively fast, lower-quality image-smoothing algorithm. Bicubic or Lanczos scaling are examples of image-smoothing algorithms that produce higher-quality output. This specification does not mandate that specific interpolation algorithms be used.

### 4.12.5.1.1 Implementation notes

This section is non-normative.

The output bitmap, when it is not directly displayed by the user agent, implementations can, instead of updating this bitmap, merely remember the sequence of drawing operations that have been applied to it until such time as the bitmap's actual data is needed (for example because of a call to `drawImage()` or the `createImageBitmap()` factory method). In many cases, this will be more memory efficient.

The bitmap of a canvas element is the one bitmap that's pretty much always going to be needed in practice. The output bitmap of a rendering context, when it has one, is always just an alias to a canvas element's bitmap.

Additional bitmaps are sometimes needed, e.g. to enable fast drawing when the canvas is being painted at a different size than its intrinsic size, or to enable double buffering so that graphics updates, like page scrolling for example, can be processed concurrently while canvas draw commands are being executed.

### 4.12.5.1.2 The canvas state

Objects that implement the `CanvasState` interface maintain a stack of drawing states. Drawing states consist of:

- The current `transformation matrix`.
- The current `clipping region`.
- The current values of the following attributes: `strokeStyle`, `fillStyle`, `globalAlpha`, `lineWidth`, `lineCap`, `lineJoin`, `miterLimit`, `lineDashOffset`, `shadowOffsetX`, `shadowOffsetY`, `shadowBlur`, `shadowColor`, `filter`, `globalCompositeOperation`, `font`, `textAlign`, `textBaseline`, `direction`, `imageSmoothingEnabled`, `imageSmoothingQuality`.
- The current `dash list`.

**Note**

The current default path and the rendering context’s bitmaps are not part of the drawing state. The current default path is persistent, and can only be reset using the `beginPath()` method. The bitmaps depend on whether and how the rendering context is bound to a canvas element.

**For web developers (non-normative)**

```javascript
context.save();
```

Pushes the current state onto the stack.
The `save()` method, when invoked, must push a copy of the current drawing state onto the drawing state stack. The `restore()` method, when invoked, must pop the top entry in the drawing state stack, and reset the drawing state it describes. If there is no saved state, then the method must do nothing.

When the user agent is to reset the rendering context to its default state, it must clear the drawing state stack and everything that drawing state consists of to initial values.

### 4.12.5.1.3 Line styles

For web developers (non-normative)

```javascript
context.lineWidth[ = value ]
styles.lineWidth[ = value ]

Returns the current line width.

Can be set, to change the line width. Values that are not finite values greater than zero are ignored.
```

```javascript
context.lineCap[ = value ]
styles.lineCap[ = value ]

Returns the current line cap style.

Can be set, to change the line cap style.

The possible line cap styles are "butt", "round", and "square". Other values are ignored.
```

```javascript
context.lineJoin[ = value ]
styles.lineJoin[ = value ]

Returns the current line join style.

Can be set, to change the line join style.

The possible line join styles are "bevel", "round", and "miter". Other values are ignored.
```

```javascript
context.miterLimit[ = value ]
styles.miterLimit[ = value ]

Returns the current miter limit ratio.

Can be set, to change the miter limit ratio. Values that are not finite values greater than zero are ignored.
```

```javascript
context.setLineDash(segments)
styles.setLineDash(segments)

Sets the current line dash pattern (as used when stroking). The argument is a list of distances for which to alternately have the line on and the line off.
```

```javascript
segments = context.getLineDash()
segments = styles.getLineDash()

Returns a copy of the current line dash pattern. The array returned will always have an even number of entries (i.e. the pattern is normalized).
```

```javascript
context.lineWidthOffset[ = value ]
styles.lineWidthOffset[ = value ]

Returns the phase offset (in the same units as the line dash pattern).

Can be set, to change the phase offset. Values that are not finite values are ignored.
```

Objects that implement the `CanvasPathDrawingStyles` interface have attributes and methods (defined in this section) that control how lines are treated by the object.

The `LineWidth` attribute gives the width of lines, in coordinate space units. On getting, it must return the current value. On setting, zero, negative, infinite, and NaN values must be ignored, leaving the value unchanged; other values must change the current value to
the new value.

When the object implementing the `CanvasPathDrawingStyles` interface is created, the `lineWidth` attribute must initially have the value 1.0.

The `lineCap` attribute defines the type of endings that UAs will place on the end of lines. The three valid values are "butt", "round", and "square".

On getting, it must return the current value. On setting, the current value must be changed to the new value.

When the object implementing the `CanvasPathDrawingStyles` interface is created, the `lineCap` attribute must initially have the value "butt".

The `lineJoin` attribute defines the type of corners that UAs will place where two lines meet. The three valid values are "bevel", "round", and "miter".

On getting, it must return the current value. On setting, the current value must be changed to the new value.

When the object implementing the `CanvasPathDrawingStyles` interface is created, the `lineJoin` attribute must initially have the value "miter".

When the `lineJoin` attribute has the value "miter", strokes use the miter limit ratio to decide how to render joins. The miter limit ratio can be explicitly set using the `miterLimit` attribute. On getting, it must return the current value. On setting, zero, negative, infinite, and NaN values must be ignored, leaving the value unchanged; other values must change the current value to the new value.

When the object implementing the `CanvasPathDrawingStyles` interface is created, the `miterLimit` attribute must initially have the value 10.0.

Each `CanvasPathDrawingStyles` object has a dash list, which is either empty or consists of an even number of non-negative numbers. Initially, the dash list must be empty.

The `setLineDash()` method, when invoked, must run these steps:

1. Let `a` be the argument.
2. If any value in `a` is not finite (e.g. an Infinity or a NaN value), or if any value is negative (less than zero), then return (without throwing an exception; user agents could show a message on a developer console, though, as that would be helpful for debugging).
3. If the number of elements in `a` is odd, then let `a` be the concatenation of two copies of `a`.
4. Let the object's dash list be `a`.

When the `getLineDash()` method is invoked, it must return a sequence whose values are the values of the object's dash list, in the same order.

It is sometimes useful to change the "phase" of the dash pattern, e.g. to achieve a "marching ants" effect. The phase can be set using the `lineDashOffset` attribute. On getting, it must return the current value. On setting, infinite and NaN values must be ignored, leaving the value unchanged; other values must change the current value to the new value.

When the object implementing the `CanvasPathDrawingStyles` interface is created, the `lineDashOffset` attribute must initially have the value 0.0.

When a user agent is to trace a path, given an object style that implements the `CanvasPathDrawingStyles` interface, it must run the following algorithm. This algorithm returns a new path.

1. Let `path` be a copy of the path being traced.
2. Prune all zero-length line segments from `path`.

647
3. Remove from path any subpaths containing no lines (i.e. subpaths with just one point).

4. Replace each point in each subpath of path other than the first point and the last point of each subpath by a join that joins the line leading to that point to the line leading out of that point, such that the subpaths all consist of two points (a starting point with a line leading out of it, and an ending point with a line leading into it), one or more lines (connecting the points and the joins), and zero or more joins (each connecting one line to another), connected together such that each subpath is a series of one or more lines with a join between each one and a point on each end.

5. Add a straight closing line to each closed subpath in path connecting the last point and the first point of that subpath; change the last point to a join (from the previously last line to the newly added closing line), and change the first point to a join (from the newly added closing line to the first line).

6. If style's dash list is empty, then jump to the step labeled convert.

7. Let pattern width be the concatenation of all the entries of style's dash list, in coordinate space units.

8. For each subpath subpath in path, run the following substeps. These substeps mutate the subpaths in path in vivo.

   1. Let subpath width be the length of all the lines of subpath, in coordinate space units.
   2. Let offset be the value of style's lineDashOffset, in coordinate space units.
   3. While offset is greater than pattern width, decrement it by pattern width.

   While offset is less than zero, increment it by pattern width.

   4. Define L to be a linear coordinate line defined along all lines in subpath, such that the start of the first line in the subpath is defined as coordinate 0, and the end of the last line in the subpath is defined as coordinate subpath width.

   5. Let position be zero minus offset.
   6. Let index be 0.

   7. Let current state be off (the other states being on and zero-on).

   8. Dash on: Let segment length be the value of style's dash list's indexth entry.

   9. Increment position by segment length.

   10. If position is greater than subpath width, then end these substeps for this subpath and start them again for the next subpath; if there are no more subpaths, then jump to the step labeled convert instead.

   11. If segment length is nonzero, then let current state be on.

   12. Increment index by one.

   13. Dash off: Let segment length be the value of style's dash list's indexth entry.

   14. Let start be the offset position on L.

   15. Increment position by segment length.

   16. If position is less than zero, then jump to the step labeled post-cut.

   17. If start is less than zero, then let start be zero.

   18. If position is greater than subpath width, then let end be the offset subpath width on L. Otherwise, let end be the offset position on L.

   19. Jump to the first appropriate step:

   ↪ If segment length is zero and current state is off
      Do nothing, just continue to the next step.

   ↪ If current state is off
      Cut the line on which end finds itself short at end and place a point there, cutting in two the subpath that it was in; remove all line segments, joins, points, and subpaths that are between start and end; and finally place a single point at start with no lines connecting to it.

      The point has a directionality for the purposes of drawing line caps (see below). The directionality is the
direction that the original line had at that point (i.e. when \( L \) was defined above).

**Otherwise**

Cut the line on which \( \text{start} \) finds itself into two at \( \text{start} \) and place a point there, cutting in two the subpath that it was in, and similarly cut the line on which \( \text{end} \) finds itself short at \( \text{end} \) and place a point there, cutting in two the subpath that it was in, and then remove all line segments, joins, points, and subpaths that are between \( \text{start} \) and \( \text{end} \).

If \( \text{start} \) and \( \text{end} \) are the same point, then this results in just the line being cut in two and two points being inserted there, with nothing being removed, unless a join also happens to be at that point, in which case the join must be removed.

20. **Post-cut:** If \( \text{position} \) is greater than \( \text{subpath width} \), then jump to the step labeled **convert**.

21. If \( \text{segment length} \) is greater than zero, then let \( \text{positioned-at-on-dash} \) be false.

22. Increment \( \text{index} \) by one. If it is equal to the number of entries in \( \text{style's} \) dash list\(^{647} \), then let \( \text{index} \) be 0.

23. Return to the step labeled **dash on**.

9. **Convert:** This is the step that converts the path to a new path that represents its stroke.

Create a new \( \text{path}^{654} \) that describes the edge of the areas that would be covered if a straight line of length equal to \( \text{style's} \) lineWidth\(^{646} \) was swept along each subpath in \( \text{path} \) while being kept at an angle such that the line is orthogonal to the path being swept, replacing each point with the end cap necessary to satisfy \( \text{style's} \) lineCap\(^{647} \) attribute as described previously and elaborated below, and replacing each join with the join necessary to satisfy \( \text{style's} \) lineJoin\(^{647} \) type, as defined below.

**Caps:** Each point has a flat edge perpendicular to the direction of the line coming out of it. This is then augmented according to the value of \( \text{style's} \) lineCap\(^{647} \). The "butt" value means that no additional line cap is added. The "round" value means that a semi-circle with the diameter equal to \( \text{style's} \) lineWidth\(^{646} \) width must additionally be placed on to the line coming out of each point. The "square" value means that a rectangle with the length of \( \text{style's} \) lineWidth\(^{646} \) width and the width of half \( \text{style's} \) lineWidth\(^{646} \) width, placed flat against the edge perpendicular to the direction of the line coming out of the point, must be added at each point.

Points with no lines coming out of them must have two caps placed back-to-back as if it was really two points connected to each other by an infinitesimally short straight line in the direction of the point's directionality (as defined above).

**Joins:** In addition to the point where a join occurs, two additional points are relevant to each join, one for each line: the two corners found half the line width away from the join point, one perpendicular to each line, each on the side furthest from the other line.

A triangle connecting these two opposite corners with a straight line, with the third point of the triangle being the join point, must be added at all joins. The \( \text{lineJoin}^{647} \) attribute controls whether anything else is rendered. The three aforementioned values have the following meanings:

- The "bevel" value means that this is all that is rendered at joins.
- The "round" value means that an arc connecting the two aforementioned corners of the join, abutting (and not overlapping) the aforementioned triangle, with the diameter equal to the line width and the origin at the point of the join, must be added at joins.
- The "miter" value means that a second triangle must (if it can given the miter length) be added at the join, with one line being the line between the two aforementioned corners, abutting the first triangle, and the other two being continuations of the outside edges of the two joining lines, as long as required to intersect without going over the miter length.

The miter length is the distance from the point where the join occurs to the intersection of the line edges on the outside of the join. The miter limit ratio is the maximum allowed ratio of the miter length to half the line width. If the miter length would cause the miter limit ratio (as set by \( \text{style's} \) miterLimit\(^{647} \) attribute) to be exceeded, then this second triangle must not be added.

The subpaths in the newly created path must be oriented such that for any point, the number of times a half-infinite straight line drawn from that point crosses a subpath is even if and only if the number of times a half-infinite straight line drawn from that same point crosses a subpath going in one direction is equal to the number of times it crosses a subpath going in the other direction.

10. Return the newly created path.
4.12.5.1.4 Text styles

Objects that implement the CanvasTextDrawingStyles interface have attributes (defined in this section) that control how text is laid out (rasterized or outlined) by the object. Such objects can also have a font style source object. For CanvasRenderingContext2D objects, this is the canvas element given by the value of the context's canvas attribute. For OffscreenCanvasRenderingContext2D objects, this is the associated OffscreenCanvas.

Font resolution for the font style source object requires a font source. This is determined for a given object implementing CanvasTextDrawingStyles by the following steps: [CSSFONTLOAD]

1. If object's font style source object is a canvas element, return the element's node document.
2. Otherwise, object's font style source object is an OffscreenCanvas object:
   1. Let global be object's relevant global object.
   2. If global is a Window object, then return global's associated Document.
   3. Assert: global implements WorkerGlobalScope.
   4. Return global.

Example

This is an example of font resolution with a regular canvas element with ID c1.

```javascript
const font = new FontFace("MyCanvasFont", "url(mycanvasfont.ttf)");
documents.fonts.add(font);

const context = document.getElementById("c1").getContext("2d");
document.fonts.ready.then(function() {
    context.font = "64px MyCanvasFont";
    context.fillText("hello", 0, 0);
});
```

In this example, the canvas will display text using mycanvasfont.ttf as its font.
Example
This is an example of how font resolution can happen using `OffscreenCanvas`. Assuming a `canvas` element with ID `c2` which is transferred to a worker like so:

```javascript
const offscreenCanvas = document.getElementById("c2").transferControlToOffscreen();
worker.postMessage(offscreenCanvas, [offscreenCanvas]);
```

Then, in the worker:

```javascript
self.onmessage = function(ev) {
  const transferredCanvas = ev.data;
  const context = transferredCanvas.getContext("2d");
  const font = new FontFace("MyFont", "url(myfont.ttf)");
  self.fonts.add(font);
  self.fonts.ready.then(function() {
    context.font = "64px MyFont";
    context.fillText("hello", 0, 0);
  });
};
```

In this example, the canvas will display a text using `myfont.ttf`. Notice that the font is only loaded inside the worker, and not in the document context.

The `font` IDL attribute, on setting, must be parsed as a CSS `<font>` value (but without supporting property-independent style sheet syntax like 'inherit'), and the resulting font must be assigned to the context, with the `line-height` component forced to 'normal', with the `font-size` component converted to CSS pixels, and with system fonts being computed to explicit values. If the new value is syntactically incorrect (including using property-independent style sheet syntax like 'inherit' or 'initial'), then it must be ignored, without assigning a new font value. [CSS][1270]

Font family names must be interpreted in the context of the `font style source object` when the font is to be used; any fonts embedded using `@font-face` or loaded using `FontFace` objects that are visible to the `font style source object` must therefore be available once they are loaded. (Each `font style source object` has a `font source`, which determines what fonts are available.) If a font is used before it is fully loaded, or if the `font style source object` does not have that font in scope at the time the font is to be used, then it must be treated as if it was an unknown font, falling back to another as described by the relevant CSS specifications. [CSSFONTS][1271] [CSSFONTLOAD][1271]

On getting, the `font` attribute must return the serialized form of the current font of the context (with no `line-height` component). [CSSOM][1271]

Example
For example, after the following statement:

```javascript
context.font = 'italic 400 12px/2 Unknown Font, sans-serif';
```

...the expression `context.font` would evaluate to the string "italic 12px "Unknown Font", sans-serif". The "400" font-weight doesn't appear because that is the default value. The line-height doesn't appear because it is forced to "normal", the default value.

When the object implementing the `CanvasTextDrawingStyles` interface is created, the font of the context must be set to 10px sans-serif. When the `font-size` component is set to lengths using percentages, 'em' or 'ex' units, or the 'larger' or 'smaller' keywords, these must be interpreted relative to the computed value of the `font-size` property of the `font style source object` at the time that the attribute is set, if it is an element. When the `font-weight` component is set to the relative values 'bolder' and 'lighter', these must be interpreted relative to the computed value of the `font-weight` property of the `font style source object` at the time that the attribute is set, if it is an element. If the computed values are undefined for a particular case (e.g. because the `font style source object` is not an element or is not being rendered), then the relative keywords must be interpreted relative to the normal-weight 10px sans-serif default.

The `textAlign` IDL attribute, on getting, must return the current value. On setting, the current value must be changed to the new value. When the object implementing the `CanvasTextDrawingStyles` interface is created, the `textAlign` attribute must initially have the value `start`.

The `textBaseline` IDL attribute, on getting, must return the current value. On setting, the current value must be changed to the new...
value. When the object implementing the `CanvasTextDrawingStyles` interface is created, the `textBaseline` attribute must initially have the value `alphabetic`.

The `direction` IDL attribute, on getting, must return the current value. On setting, the current value must be changed to the new value. When the object implementing the `CanvasTextDrawingStyles` interface is created, the `direction` attribute must initially have the value "inherit".

The `textAlign` attribute's allowed keywords are as follows:

- **start**
  - Align to the start edge of the text (left side in left-to-right text, right side in right-to-left text).

- **end**
  - Align to the end edge of the text (right side in left-to-right text, left side in right-to-left text).

- **left**
  - Align to the left.

- **right**
  - Align to the right.

- **center**
  - Align to the center.

The `textBaseline` attribute's allowed keywords correspond to alignment points in the font:

- **top**
  - The top of the em square

- **hanging**
  - The hanging baseline

- **middle**
  - The middle of the em square

- **alphabetic**
  - The alphabetic baseline

- **ideographic**
  - The ideographic baseline

The keywords map to these alignment points as follows:

- **top**
  - The top of the em square

- **hanging**
  - The hanging baseline

- **middle**
  - The middle of the em square

- **alphabetic**
  - The alphabetic baseline

- **ideographic**
  - The ideographic baseline
The bottom of the em square

The **direction** attribute’s allowed keywords are as follows:

- **ltr**
  - Treat input to the text preparation algorithm as left-to-right text.

- **rtl**
  - Treat input to the text preparation algorithm as right-to-left text.

- **inherit**
  - Default to the directionality of the canvas element or Document as appropriate.

The text preparation algorithm is as follows. It takes as input a string `text`, a `CanvasTextDrawingStyles` object `target`, and an optional length `maxWidth`. It returns an array of glyph shapes, each positioned on a common coordinate space, a physical alignment whose value is one of left, right, and center, and an inline box. (Most callers of this algorithm ignore the physical alignment and the inline box.)

1. If `maxWidth` was provided but is less than or equal to zero or equal to NaN, then return an empty array.
2. Replace all ASCII whitespace in `text` with U+0020 SPACE characters.
3. Let `font` be the current font of `target`, as given by that object's `font` attribute.
4. Apply the appropriate step from the following list to determine the value of `direction`:
   - If the target object’s `direction` attribute has the value "ltr"
     Let direction be 'ltr'.
   - If the target object’s `direction` attribute has the value "rtl"
     Let direction be 'rtl'.
   - If the target object’s font style source object is an element
     Let direction be the directionality of the target object’s font style source.
   - If the target object’s font style source object is a `Document` with a non-null document element
     Let direction be the directionality of the target object’s font style source's document element.
   - Otherwise
     Let direction be 'ltr'.
5. Form a hypothetical infinitely-wide CSS line box containing a single inline box containing the text `text`, with all the properties at their initial values except the 'font' property of the inline box set to font, the 'direction' property of the inline box set to `direction`, and the 'white-space' property set to 'pre'. [CSS]
6. If `maxWidth` was provided and the hypothetical width of the inline box in the hypothetical line box is greater than `maxWidth` CSS pixels, then change font to have a more condensed font (if one is available or if a reasonably readable one can be synthesized by applying a horizontal scale factor to the font) or a smaller font, and return to the previous step.
7. The anchor point is a point on the inline box, and the physical alignment is one of the values left, right, and center. These variables are determined by the `textAlign` and `textBaseline` values as follows:

Horizontal position:
   - If `textAlign` is `left`
   - If `textAlign` is `start` and `direction` is 'ltr'
   - If `textAlign` is `end` and `direction` is 'rtl'
     Let the anchor point's horizontal position be the left edge of the inline box, and let physical alignment be left.
   - If `textAlign` is `right`
   - If `textAlign` is `end` and `direction` is 'ltr'
   - If `textAlign` is `start` and `direction` is 'rtl'
     Let the anchor point's horizontal position be the right edge of the inline box, and let physical alignment be right.
If `textAlign` is `center`

Let the anchor point's horizontal position be half way between the left and right edges of the inline box, and let physical alignment be center.

Vertical position:

If `textBaseline` is `top`

Let the anchor point's vertical position be the top of the em box of the first available font of the inline box.

If `textBaseline` is `hanging`

Let the anchor point's vertical position be the hanging baseline of the first available font of the inline box.

If `textBaseline` is `middle`

Let the anchor point's vertical position be half way between the bottom and the top of the em box of the first available font of the inline box.

If `textBaseline` is `ideographic`

Let the anchor point's vertical position be the ideographic-under baseline of the first available font of the inline box.

If `textBaseline` is `bottom`

Let the anchor point's vertical position be the bottom of the em box of the first available font of the inline box.

8. Let result be an array constructed by iterating over each glyph in the inline box from left to right (if any), adding to the array, for each glyph, the shape of the glyph as it is in the inline box, positioned on a coordinate space using CSS pixels with its origin is at the anchor point.

9. Return result, physical alignment, and the inline box.

4.12.5.1.5 Building paths

Objects that implement the CanvasPath interface have a path. A path has a list of zero or more subpaths. Each subpath consists of a list of one or more points, connected by straight or curved line segments, and a flag indicating whether the subpath is closed or not. A closed subpath is one where the last point of the subpath is connected to the first point of the subpath by a straight line. Subpaths with only one point are ignored when painting the path.

Paths have a need new subpath flag. When this flag is set, certain APIs create a new subpath rather than extending the previous one. When a path is created, its need new subpath flag must be set.

When an object implementing the CanvasPath interface is created, its path must be initialized to zero subpaths.

**For web developers (non-normative)**

```
context . moveTo(x, y)

path . moveTo(x, y)

  Creates a new subpath with the given point.

context . closePath()

path . closePath()

  Marks the current subpath as closed, and starts a new subpath with a point the same as the start and end of the newly closed subpath.

context . lineTo(x, y)

path . lineTo(x, y)

  Adds the given point to the current subpath, connected to the previous one by a straight line.

context . quadraticCurveTo(cpx, cpy, x, y)

path . quadraticCurveTo(cpx, cpy, x, y)

  Adds the given point to the current subpath, connected to the previous one by a quadratic Bézier curve with the given control point.
```
context.bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)

Adds the given point to the current subpath, connected to the previous one by a cubic Bézier curve with the given control points.

context.arcTo(x1, y1, x2, y2, radius)

Adds an arc with the given control points and radius to the current subpath, connected to the previous point by a straight line. Throws an "IndexSizeError" DOMException if the given radius is negative.
The following methods allow authors to manipulate the paths of objects implementing the `CanvasPath` interface.

For objects implementing the `CanvasDrawPath` and `CanvasTransform` interfaces, the points passed to the methods, and the resulting lines added to the `current default path` by these methods, must be transformed according to the `current transformation`.
The **moveTo**(x, y) method, when invoked, must run these steps:

1. If either of the arguments are infinite or NaN, then return.
2. Create a new subpath with the specified point as its first (and only) point.

When the user agent is to **ensure there is a subpath** for a coordinate (x, y) on a path, the user agent must check to see if the path has its need new subpath flag set. If it does, then the user agent must create a new subpath with the point (x, y) as its first (and only) point, as if the moveTo() method had been called, and must then unset the path’s need new subpath flag.

The **closePath**() method, when invoked, must do nothing if the object’s path has no subpaths. Otherwise, it must mark the last subpath as closed, create a new subpath whose first point is the same as the previous subpath’s first point, and finally add this new subpath to the path.

New points and the lines connecting them are added to subpaths using the methods described below. In all cases, the methods only modify the last subpath in the object’s path.

The **lineTo**(x, y) method, when invoked, must run these steps:

1. If either of the arguments are infinite or NaN, then return.
2. If the object’s path has no subpaths, then **ensure there is a subpath** for (x, y).
3. Otherwise, connect the last point in the subpath to the given point (x, y) using a straight line, and then add the given point (x, y) to the subpath.

The **quadraticCurveTo**(cpx, cpy, x, y) method, when invoked, must run these steps:

1. If any of the arguments are infinite or NaN, then return.
2. **Ensure there is a subpath** for (cpx, cpy).
3. Connect the last point in the subpath to the given point (x, y) using a quadratic Bézier curve with control point (cpx, cpy).
4. Add the given point (x, y) to the subpath.

The **bezierCurveTo**(cp1x, cp1y, cp2x, cp2y, x, y) method, when invoked, must run these steps:

1. If any of the arguments are infinite or NaN, then return.
2. **Ensure there is a subpath** for (cp1x, cp1y).
3. Connect the last point in the subpath to the given point (x, y) using a cubic Bézier curve with control points (cp1x, cp1y) and (cp2x, cp2y).
4. Add the point (x, y) to the subpath.

The **arcTo**(x1, y1, x2, y2, radius) method, when invoked, must run these steps:

1. If any of the arguments are infinite or NaN, then return.
2. **Ensure there is a subpath** for (x1, y1).
3. If radius is negative, then throw an "IndexSizeError" DOMException.
4. Let the point (x0, y0) be the last point in the subpath, transformed by the inverse of the current transformation matrix (so that it is in the same coordinate system as the points passed to the method).
5. If the point \((x_0, y_0)\) is equal to the point \((x_1, y_1)\), or if the point \((x_1, y_1)\) is equal to the point \((x_2, y_2)\), or if \(radius\) is zero, then add the point \((x_1, y_1)\) to the subpath, and connect that point to the previous point \((x_0, y_0)\) by a straight line.

6. Otherwise, if the points \((x_0, y_0), (x_1, y_1),\) and \((x_2, y_2)\) all lie on a single straight line, then add the point \((x_1, y_1)\) to the subpath, and connect that point to the previous point \((x_0, y_0)\) by a straight line.

7. Otherwise, let The Arc be the shortest arc given by circumference of the circle that has radius \(radius\), and that has one point tangent to the half-infinite line that crosses the point \((x_0, y_0)\) and ends at the point \((x_1, y_1)\), and that has a different point tangent to the half-infinite line that ends at the point \((x_1, y_1)\) and crosses the point \((x_2, y_2)\). The points at which this circle touches these two lines are called the start and end tangent points respectively. Connect the point \((x_0, y_0)\) to the start tangent point by a straight line, adding the start tangent point to the subpath, and then connect the start tangent point to the end tangent point by The Arc, adding the end tangent point to the subpath.

The \(arc(x, y, radius, startAngle, endAngle, counterclockwise)\) method, when invoked, must run the ellipse method steps with this, \(x, y, radius, radius, 0, startAngle, endAngle,\) and \(counterclockwise\).

Note
This makes it equivalent to \(ellipse())\) except that both radii are equal and rotation is 0.

The \(ellipse(x, y, radiusX, radiusY, rotation, startAngle, endAngle, counterclockwise)\) method, when invoked, must run the ellipse method steps with this, \(x, y, radiusX, radiusY, rotation, startAngle, endAngle,\) and \(counterclockwise\).

The ellipse method steps, given \(canvasPath, x, y, radiusX, radiusY, rotation, startAngle, endAngle,\) and \(counterclockwise\), are:

1. If any of the arguments are infinite or NaN, then return.
2. If either \(radiusX\) or \(radiusY\) are negative, then throw an "IndexSizeError" DOMException.
3. If \(canvasPath\)'s path has any subpaths, then add a straight line from the last point in the subpath to the start point of the arc.
4. Add the start and end points of the arc to the subpath, and connect them with an arc. The arc and its start and end points are defined as follows:

   Consider an ellipse that has its origin at \((x, y)\), that has a major-axis radius \(radiusX\) and a minor-axis radius \(radiusY\), and that is rotated about its origin such that its semi-major axis is inclined \(rotation\) radians clockwise from the x-axis.

   If \(counterclockwise\) is false and \(endAngle-startAngle\) is equal to or greater than \(2\pi\), or, if \(counterclockwise\) is true and \(startAngle-endAngle\) is equal to or greater than \(2\pi\), then the arc is the whole circumference of this ellipse, and the point at \(startAngle\) along this circle's circumference, measured in radians clockwise from the ellipse's semi-major axis, acts as both the start point and the end point.

   Otherwise, the points at \(startAngle\) and \(endAngle\) along this circle's circumference, measured in radians clockwise from the ellipse's semi-major axis, are the start and end points respectively, and the arc is the path along the circumference of this ellipse from the start point to the end point, going counterclockwise if \(counterclockwise\) is true, and clockwise otherwise.

   Since the points are on the ellipse, as opposed to being simply angles from zero, the arc can never cover an angle greater than \(2\pi\) radians.

   Note
Even if the arc covers the entire circumference of the ellipse and there are no other points in the subpath, the path is not closed unless the closePath() method is appropriately invoked.

The \(rect(x, y, w, h)\) method, when invoked, must run these steps:

1. If any of the arguments are infinite or NaN, then return.
2. Create a new subpath containing just the four points \((x, y), (x+w, y), (x+w, y+h), (x, y+h)\), in that order, with those four points connected by straight lines.
3. Mark the subpath as closed.
4. Create a new subpath with the point \((x, y)\) as the only point in the subpath.
Path2D objects can be used to declare paths that are then later used on objects implementing the CanvasDrawPath interface. In addition to many of the APIs described in earlier sections, Path2D objects have methods to combine paths, and to add text to paths.

For web developers (non-normative)

\[\text{path} = \text{new Path2D()}\]

Creates a new empty Path2D object.

\[\text{path} = \text{new Path2D(path)}\]

When `path` is a Path2D object, returns a copy.
When `path` is a string, creates the path described by the argument, interpreted as SVG path data. [SVG]

\[\text{path}\!.\text{addPath(path [, transform ])}\]

Adds to the path the path given by the argument.

The Path2D constructor, when invoked, must run these steps:

1. Let `output` be a new Path2D object.
2. If `path` is not given, then return `output`.
3. If `path` is a Path2D object, then add all subpaths of `path` to `output` and return `output`. (In other words, it returns a copy of the argument.)
4. Let `svgPath` be the result of parsing and interpreting `path` according to SVG 2's rules for path data. [SVG]

Note

The resulting path could be empty. SVG defines error handling rules for parsing and applying path data.

5. Let \((x, y)\) be the last point in `svgPath`.
6. Add all the subpaths, if any, from `svgPath` to `output`.
7. Create a new subpath in `output` with \((x, y)\) as the only point in the subpath.
8. Return `output`.

The addPath method, when invoked on a Path2D object `a`, must run these steps:

1. If the Path2D object `b` has no subpaths, then return.
2. Let `matrix` be the result of creating a DOMMatrix from the 2D dictionary `transform`.
3. If one or more of matrix's \(m11\) element, \(m12\) element, \(m21\) element, \(m22\) element, \(m41\) element, or \(m42\) element are infinite or NaN, then return.
4. Create a copy of all the subpaths in `b`. Let this copy be known as `c`.
5. Transform all the coordinates and lines in `c` by the transform matrix `matrix`.
6. Let \((x, y)\) be the last point in the last subpath of `c`.
7. Add all the subpaths in `c` to `a`.
8. Create a new subpath in `a` with \((x, y)\) as the only point in the subpath.

4.12.5.1.7 Transformations

Objects that implement the CanvasTransform interface have a current transformation matrix, as well as methods (described in this section) to manipulate it. When an object implementing the CanvasTransform interface is created, its transformation matrix must be initialized to the identity matrix.

The current transformation matrix is applied to coordinates when creating the current default path, and when painting text,
shapes, and Path2D objects, on objects implementing the CanvasTransform interface.

The transformations must be performed in reverse order.

**Note**

For instance, if a scale transformation that doubles the width is applied to the canvas, followed by a rotation transformation that rotates drawing operations by a quarter turn, and a rectangle twice as wide as it is tall is then drawn on the canvas, the actual result will be a square.

For web developers (non-normative)

```plaintext
context.scale(x, y)
Changes the current transformation matrix to apply a scaling transformation with the given characteristics.

context.rotate(angle)
Changes the current transformation matrix to apply a rotation transformation with the given characteristics. The angle is in radians.

context.translate(x, y)
Changes the current transformation matrix to apply a translation transformation with the given characteristics.

context.transform(a, b, c, d, e, f)
Changes the current transformation matrix to apply the matrix given by the arguments as described below.

matrix = context.getTransform()
Returns a copy of the current transformation matrix, as a newly created DOMMatrix object.

context.setTransform(a, b, c, d, e, f)
Changes the current transformation matrix to the matrix given by the arguments as described below.

context.setTransform(transform)
Changes the current transformation matrix to the matrix represented by the passed DOMMatrix2DInit dictionary.

context.resetTransform()
Changes the current transformation matrix to the identity matrix.
```

The `scale(x, y)` method, when invoked, must run these steps:

1. If either of the arguments are infinite or NaN, then return.
2. Add the scaling transformation described by the arguments to the current transformation matrix. The `x` argument represents the scale factor in the horizontal direction and the `y` argument represents the scale factor in the vertical direction. The factors are multiples.

The `rotate(angle)` method, when invoked, must run these steps:

1. If `angle` is infinite or NaN, then return.
2. Add the rotation transformation described by the argument to the current transformation matrix. The `angle` argument represents a clockwise rotation angle expressed in radians.

The `translate(x, y)` method, when invoked, must run these steps:

1. If either of the arguments are infinite or NaN, then return.
2. Add the translation transformation described by the arguments to the current transformation matrix. The `x` argument represents the translation distance in the horizontal direction and the `y` argument represents the translation distance in the vertical direction. The arguments are in coordinate space units.

The `transform(a, b, c, d, e, f)` method, when invoked, must run these steps:

1. If any of the arguments are infinite or NaN, then return.
2. Replace the current transformation matrix with the result of multiplying the current transformation matrix with the matrix described by:
The `getTransform()` method, when invoked, must return a newly created `DOMMatrix` representing a copy of the current transformation matrix of the context.

**Note**

The arguments `a, b, c, d, e, and f` are sometimes called `m11, m12, m21, m22, dx, and dy or m11, m21, m12, m22, dx, and dy`. Care ought to be taken in particular with the order of the second and third arguments (`b` and `c`) as their order varies from API to API and APIs sometimes use the notation `m12/m21` and sometimes `m21/m12` for those positions.

This returned object is not live, so updating it will not affect the current transformation matrix, and updating the current transformation matrix will not affect an already returned `DOMMatrix`.

The `setTransform(a, b, c, d, e, f)` method, when invoked, must run these steps:

1. If any of the arguments are infinite or NaN, then return.
2. Reset the current transformation matrix to the identity matrix.
3. Invoke the `transform(a, b, c, d, e, f)` method with the same arguments.

The `setTransform(transform)` method, when invoked, must run these steps:

1. Let `matrix` be the result of creating a `DOMMatrix` from the 2D dictionary `transform`.
2. If one or more of `matrix`'s `m11 element, m12 element, m21 element, m22 element, m41 element, or m42 element` are infinite or NaN, then return.
3. Reset the current transformation matrix to `matrix`.

The `resetTransform()` method, when invoked, must reset the current transformation matrix to the identity matrix.

**Note**

Given a matrix of the form created by the `transform()` and `setTransform()` methods, i.e.,

<table>
<thead>
<tr>
<th>a</th>
<th>c</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>d</td>
<td>f</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

the resulting transformed coordinates after transform matrix multiplication will be

\[
\begin{align*}
x_{\text{new}} &= ax + cy + e \\
y_{\text{new}} &= bx + dy + f
\end{align*}
\]

### 4.12.5.1.8 Image sources for 2D rendering contexts

Some methods on the `CanvasDrawImage` and `CanvasFillStrokeStyles` interfaces take the union type `CanvasImageSource` as an argument.

This union type allows objects implementing any of the following interfaces to be used as image sources:

- `HTMLOrSVGImageElement` (img or SVG image elements)
- `HTMLVideoElement` (video elements)
- `HTMLCanvasElement` (canvas elements)
- `ImageBitmap`
To check the usability of the image argument, where image is a `CanvasImageSource` object, run these steps:

1. Switch on image:
   - `HTMLOrSVGImageElement`
     1. If image's `current request`'s `state` is `broken`, then throw an "InvalidStateError" `DOMException`.
     2. If image is not `fully decodable`, then return `bad`.
     3. If image has an `intrinsic width` or `intrinsic height` (or both) equal to zero, then return `bad`.
   - `HTMLVideoElement`
     1. If image's `readyState` attribute is either `HAVE NOTHING` or `HAVE METADATA`, then return `bad`.
   - `HTMLCanvasElement`
     1. `OffscreenCanvas`
     2. If image has either a horizontal dimension or a vertical dimension equal to zero, then throw an "InvalidStateError" `DOMException`.
   - `ImageBitmap`
     1. If image's `[[Detached]]` internal slot value is set to true, then throw an "InvalidStateError" `DOMException`.

2. Return `good`.

When a `CanvasImageSource` object represents an `HTMLOrSVGImageElement`, the element's image must be used as the source image.

Specifically, when a `CanvasImageSource` object represents an animated image in an `HTMLOrSVGImageElement`, the user agent must use 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, when rendering the image for `CanvasRenderingContext2D APIs`.

When a `CanvasImageSource` object represents an `HTMLVideoElement`, then the frame at the `current playback position` when the method with the argument is invoked must be used as the source image when rendering the image for `CanvasRenderingContext2D APIs`, and the source image's dimensions must be the `intrinsic width` and `intrinsic height` of the `media resource` (i.e. after any aspect-ratio correction has been applied).

When a `CanvasImageSource` object represents an `HTMLCanvasElement`, the element's bitmap must be used as the source image.

When a `CanvasImageSource` object represents an element that is `being rendered` and that element has been resized, the original image data of the source image must be used, not the image as it is rendered (e.g. `width` and `height` attributes on the source element have no effect on how the object is interpreted when rendering the image for `CanvasRenderingContext2D APIs`).

When a `CanvasImageSource` object represents an `ImageBitmap`, the object's bitmap image data must be used as the source image.

An object image is not origin-clean if, switching on image:

- `HTMLOrSVGImageElement`
- `HTMLVideoElement`
  1. image's `origin` is not `same origin` with `entry settings object`'s `origin`.
- `HTMLCanvasElement`
- `ImageBitmap`
  1. image's bitmap's `origin-clean` flag is false.
For web developers (non-normative)

context.fillStyle [ = value ]

Returns the current style used for filling shapes.

Can be set, to change the fill style.

The style can be either a string containing a CSS color, or a CanvasGradient or CanvasPattern object. Invalid values are ignored.

canvasContext.strokeStyle [ = value ]

Returns the current style used for stroking shapes.

Can be set, to change the stroke style.

The style can be either a string containing a CSS color, or a CanvasGradient or CanvasPattern object. Invalid values are ignored.

Objects that implement the CanvasFillStrokeStyles interface have attributes and methods (defined in this section) that control how shapes are treated by the object.

The fillStyle attribute represents the color or style to use inside shapes, and the strokeLine attribute represents the color or style to use for the lines around the shapes.

Both attributes can be either strings, CanvasGradient's, or CanvasPattern's. On setting, strings must be parsed with this canvas element and the color assigned, and CanvasGradient and CanvasPattern objects must be assigned themselves. If parsing the value results in failure, then it must be ignored, and the attribute must retain its previous value. If the new value is a CanvasPattern object that is marked as not origin-clean, then the CanvasRenderingContext2D's origin-clean flag must be set to false.

When set to a CanvasPattern or CanvasGradient object, the assignment is live, meaning that changes made to the object after the assignment do affect subsequent stroking or filling of shapes.

On getting, if the value is a color, then the serialization of the color must be returned. Otherwise, if it is not a color but a CanvasGradient or CanvasPattern, then the respective object must be returned. (Such objects are opaque and therefore only useful for assigning to other attributes or for comparison to other gradients or patterns.)

The serialization of a color for a color value is a string, computed as follows: if it has alpha equal to 1.0, then the string is a lowercase six-digit hex value, prefixed with a "#" character (U+0023 NUMBER SIGN), with the first two digits representing the red component, the next two digits representing the green component, and the last two digits representing the blue component, the digits being ASCII lower hex digits. Otherwise, the color value has alpha less than 1.0, and the string is the color value in the CSS rgba() functional-notation format: the literal string "rgba" (U+0072 U+0067 U+0062 U+0061) followed by a U+0028 LEFT PARENTHESIS, a base-ten integer in the range 0-255 representing the red component (using ASCII digits in the shortest form possible), a literal U+002C COMMA and U+0020 SPACE, an integer for the green component, a comma and a space, an integer for the blue component, another comma and space, a U+0030 DIGIT ZERO, if the alpha value is greater than zero then a U+002E FULL STOP (representing the decimal point), if the alpha value is greater than zero then one or more ASCII digits representing the fractional part of the alpha value, and finally a U+0029 RIGHT PARENTHESIS. User agents must express the fractional part of the alpha value, if any, with the level of precision necessary for the alpha value, when reparsed, to be interpreted as the same alpha value.

When the context is created, the fillStyle and strokeLine attributes must initially have the string value #000000.

When the value is a color, it must not be affected by the transformation matrix when used to draw on bitmaps.

There are two types of gradients, linear gradients and radial gradients, both represented by objects implementing the opaque CanvasGradient interface.

Once a gradient has been created (see below), stops are placed along it to define how the colors are distributed along the gradient. The color of the gradient at each stop is the color specified for that stop. Between each such stop, the colors and the alpha component must be linearly interpolated over the RGBA space without premultiplying the alpha value to find the color to use at that offset. Before the first stop, the color must be the color of the first stop. After the last stop, the color must be the color of the last stop. When there are no stops, the gradient is transparent black.
gradient.addColorStop(offset, color)

Adds a color stop with the given color to the gradient at the given offset. 0.0 is the offset at one end of the gradient, 1.0 is the offset at the other end.

Throws an "IndexSizeError" DOMException if the offset is out of range. Throws a "SyntaxError" DOMException if the color cannot be parsed.

gradient = context.createLinearGradient(x0, y0, x1, y1)

Returns a CanvasGradient object that represents a linear gradient that paints along the line given by the coordinates represented by the arguments.

gradient = context.createRadialGradient(x0, y0, r0, x1, y1, r1)

Returns a CanvasGradient object that represents a radial gradient that paints along the cone given by the circles represented by the arguments.

If either of the radii are negative, throws an "IndexSizeError" DOMException exception.

The addColorStop(offset, color) method on the CanvasGradient, when invoked, must run these steps:

1. If the offset is less than 0 or greater than 1, then throw an "IndexSizeError" DOMException.
2. Let parsed color be the result of parsing color.

   Note

   No element is passed to the parser because CanvasGradient objects are canvas-neutral — a CanvasGradient object created by one canvas can be used by another, and there is therefore no way to know which is the "element in question" at the time that the color is specified.

3. If parsed color is failure, throw a "SyntaxError" DOMException.
4. Place a new stop on the gradient, at offset offset relative to the whole gradient, and with the color parsed color.

   If multiple stops are added at the same offset on a gradient, then they must be placed in the order added, with the first one closest to the start of the gradient, and each subsequent one infinitesimally further along towards the end point (in effect causing all but the first and last stop added at each point to be ignored).

The createLinearGradient(x0, y0, x1, y1) method takes four arguments that represent the start point (x0, y0) and end point (x1, y1) of the gradient. The method, when invoked, must return a linear CanvasGradient initialized with the specified line.

Linear gradients must be rendered such that all points on a line perpendicular to the line that crosses the start and end points have the color at the point where those two lines cross (with the colors coming from the interpolation and extrapolation described above). The points in the linear gradient must be transformed as described by the current transformation matrix when rendering.

If x0 = x1 and y0 = y1, then the linear gradient must paint nothing.

The createRadialGradient(x0, y0, r0, x1, y1, r1) method takes six arguments, the first three representing the start circle with origin (x0, y0) and radius r0, and the last three representing the end circle with origin (x1, y1) and radius r1. The values are in coordinate space units. If either of r0 or r1 are negative, then an "IndexSizeError" DOMException must be thrown. Otherwise, the method, when invoked, must return a radial CanvasGradient initialized with the two specified circles.

Radial gradients must be rendered by following these steps:

1. If x0 = x1 and y0 = y1 and r0 = r1, then the radial gradient must paint nothing. Return.
2. Let x(ω) = (x1-x0)ω + x0
   Let y(ω) = (y1-y0)ω + y0
   Let r(ω) = (r1-r0)ω + r0
   Let the color at ω be the color at that position on the gradient (with the colors coming from the interpolation and extrapolation described above).
3. For all values of ω where r(ω) > 0, starting with the value of ω nearest to positive infinity and ending with the value of ω nearest to negative infinity, draw the circumference of the circle with radius r(ω) at position (x(ω), y(ω)), with the color at ω, but only painting on the parts of the bitmap that have not yet been painted on by earlier circles in this step for this rendering.
of the gradient.

**Note**

This effectively creates a cone, touched by the two circles defined in the creation of the gradient, with the part of the cone before the start circle (0.0) using the color of the first offset, the part of the cone after the end circle (1.0) using the color of the last offset, and areas outside the cone untouched by the gradient (transparent black).

The resulting radial gradient must then be transformed as described by the current transformation matrix when rendering.

Gradients must be painted only where the relevant stroking or filling effects requires that they be drawn.

Patterns are represented by objects implementing the opaque `CanvasPattern` interface.

```javascript
pattern = context.createPattern(image, repetition)
```

Returns a `CanvasPattern` object that uses the given image and repeats in the direction(s) given by the `repetition` argument.

The allowed values for `repetition` are `repeat` (both directions), `repeat-x` (horizontal only), `repeat-y` (vertical only), and `no-repeat` (neither). If the `repetition` argument is empty, the value `repeat` is used.

If the image isn't yet fully decoded, then nothing is drawn. If the image is a canvas with no data, throws an "InvalidStateError" DOMException.

```javascript
pattern.setTransform(transform)
```

Sets the transformation matrix that will be used when rendering the pattern during a fill or stroke painting operation.

The `createPattern(image, repetition)` method, when invoked, must run these steps:

1. Let `usability` be the result of checking the usability of image.
2. If `usability` is `bad`, then return null.
3. Assert: `usability` is `good`.
4. If `repetition` is the empty string, then set it to "repeat".
5. If `repetition` is not identical to one of "repeat", "repeat-x", "repeat-y", or "no-repeat", then throw a "SyntaxError" DOMException.
6. Let `pattern` be a new `CanvasPattern` object with the image `image` and the repetition behavior given by `repetition`.
7. If `image` is not origin-clean, then mark `pattern` as `not origin-clean`.
8. Return `pattern`.

Modifying the `image` used when creating a `CanvasPattern` object after calling the `createPattern()` method must not affect the pattern(s) rendered by the `CanvasPattern` object.

Patterns have a transformation matrix, which controls how the pattern is used when it is painted. Initially, a pattern's transformation matrix must be the identity matrix.

The `setTransform(transform)` method, when invoked, must run these steps:

1. Let `matrix` be the result of creating a DOMMatrix from the 2D dictionary `transform`.
2. If one or more of `matrix`'s `m11` element, `m12` element, `m21` element, `m22` element, `m41` element, or `m42` element are infinite or NaN, then return.
3. Reset the pattern's transformation matrix to `matrix`.

When a pattern is to be rendered within an area, the user agent must run the following steps to determine what is rendered:

1. Create an infinite transparent black bitmap.
2. Place a copy of the image on the bitmap, anchored such that its top left corner is at the origin of the coordinate space, with one coordinate space unit per CSS pixel of the image, then place repeated copies of this image horizontally to the left and
right, if the repetition behavior is "repeat-x", or vertically up and down, if the repetition behavior is "repeat-y", or in all four directions all over the bitmap, if the repetition behavior is "repeat".

If the original image data is a bitmap image, then the value painted at a point in the area of the repetitions is computed by filtering the original image data. When scaling up, if the `imageSmoothingEnabled` attribute is set to false, then the image must be rendered using nearest-neighbor interpolation. Otherwise, the user agent may use any filtering algorithm (for example bilinear interpolation or nearest-neighbor). User agents which support multiple filtering algorithms may use the value of the `imageSmoothingQuality` attribute to guide the choice of filtering algorithm. When such a filtering algorithm requires a pixel value from outside the original image data, it must instead use the value from wrapping the pixel's coordinates to the original image's dimensions. (That is, the filter uses 'repeat' behavior, regardless of the value of the pattern's repetition behavior.)

3. Transform the resulting bitmap according to the pattern's transformation matrix.
4. Transform the resulting bitmap again, this time according to the current transformation matrix.
5. Replace any part of the image outside the area in which the pattern is to be rendered with transparent black.
6. The resulting bitmap is what is to be rendered, with the same origin and same scale.

If a radial gradient or repeated pattern is used when the transformation matrix is singular, then the resulting style must be transparent black (otherwise the gradient or pattern would be collapsed to a point or line, leaving the other pixels undefined). Linear gradients and solid colors always define all points even with singular transformation matrices.

### 4.12.5.1.10 Drawing rectangles to the bitmap

Objects that implement the CanvasRect interface provide the following methods for immediately drawing rectangles to the bitmap. The methods each take four arguments; the first two give the x and y coordinates of the top left of the rectangle, and the second two give the width w and height h of the rectangle, respectively.

The current transformation matrix must be applied to the following four coordinates, which form the path that must then be closed to get the specified rectangle: \((x, y), (x+w, y), (x+w, y+h), (x, y+h)\).

Shapes are painted without affecting the current default path, and are subject to the clipping region, and, with the exception of `clearRect()`, also shadow effects, global alpha, and global composition operators.

#### For web developers (non-normative)

**context**. `clearRect(x, y, w, h)`

Clears all pixels on the bitmap in the given rectangle to transparent black.

**context**. `fillRect(x, y, w, h)`

Paints the given rectangle onto the bitmap, using the current fill style.

**context**. `strokeRect(x, y, w, h)`

Paints the box that outlines the given rectangle onto the bitmap, using the current stroke style.

The `clearRect(x, y, w, h)` method, when invoked, must run these steps:

1. If any of the arguments are infinite or NaN, then return.
2. Let `pixels` be the set of pixels in the specified rectangle that also intersect the current clipping region.
3. Clear the pixels in `pixels` to transparent black, erasing any previous image.

**Note**

*If either height or width are zero, this method has no effect, since the set of pixels would be empty.*

The `fillRect(x, y, w, h)` method, when invoked, must run these steps:

1. If any of the arguments are infinite or NaN, then return.
2. If either w or h are zero, then return.
3. Paint the specified rectangular area using the `fillStyle`.

The `strokeRect(x, y, w, h)` method, when invoked, must run these steps:

1. If any of the arguments are infinite or NaN, then return.

2. Take the result of `tracing the path` described below, using the `CanvasPathDrawingStyles` interface's line styles, and fill it with the `strokeStyle`.

If both \(w\) and \(h\) are zero, the path has a single subpath with just one point \((x, y)\), and no lines, and this method thus has no effect (the `trace a path` algorithm returns an empty path in that case).

If just one of either \(w\) or \(h\) is zero, then the path has a single subpath consisting of two points, with coordinates \((x, y)\) and \((x+w, y+h)\), in that order, connected by a single straight line.

Otherwise, the path has a single subpath consisting of four points, with coordinates \((x, y)\), \((x+w, y)\), \((x+w, y+h)\), and \((x, y+h)\), connected to each other in that order by straight lines.

### 4.12.5.1.11 Drawing text to the bitmap

For web developers (non-normative)

```javascript
context.fillText(text, x, y[, maxWidth])
context.strokeText(text, x, y[, maxWidth])
```

Fills or strokes (respectively) the given text at the given position. If a maximum width is provided, the text will be scaled to fit that width if necessary.

```javascript
metrics = context.measureText(text)
```

Returns a `TextMetrics` object with the metrics of the given text in the current font.

- `metrics.width`
- `metrics.actualBoundingBoxLeft`
- `metrics.actualBoundingBoxRight`
- `metrics.fontBoundingBoxAscent`
- `metrics.fontBoundingBoxDescent`
- `metrics.actualBoundingBoxAscent`
- `metrics.actualBoundingBoxDescent`
- `metrics.emHeightAscent`
- `metrics.emHeightDescent`
- `metrics.hangingBaseline`
- `metrics.ideographicBaseline`

Returns the measurement described below.

Objects that implement the `CanvasText` interface provide the following methods for rendering text.

The `fillText()` and `strokeText()` methods take three or four arguments, `text`, `x`, `y`, and optionally `maxWidth`, and render the given text at the given \((x, y)\) coordinates ensuring that the text isn't wider than `maxWidth` if specified, using the current `font`, `textAlign`, and `textBaseline` values. Specifically, when the methods are invoked, the user agent must run these steps:

1. If any of the arguments are infinite or NaN, then return.

2. Run the `text preparation algorithm`, passing it `text`, the object implementing the `CanvasText` interface, and, if the `maxWidth` argument was provided, that argument. Let `glyphs` be the result.

3. Move all the shapes in `glyphs` to the right by \(x\) CSS pixels and down by \(y\) CSS pixels.

4. Paint the shapes given in `glyphs`, as transformed by the `current transformation matrix`, with each CSS pixel in the coordinate space of `glyphs` mapped to one coordinate space unit.

   For `fillText()`, `fillStyle` must be applied to the shapes and `strokeStyle` must be ignored. For
strokeText(), the reverse holds: strokeStyle must be applied to the result of tracing, the shapes using the object implementing the CanvasText interface for the line styles, and fillStyle must be ignored.

These shapes are painted without affecting the current path, and are subject to shadow effects, global alpha, the clipping region, and global composition operators.

The measureText() method takes one argument, text. When the method is invoked, the user agent must run the text preparation algorithm, passing it text and the object implementing the CanvasText interface, and then using the returned inline box must return a new TextMetrics object with members behaving as described in the following list:

width attribute
The width of that inline box, in CSS pixels. (The text's advance width.)

actualBoundingBoxLeft attribute
The distance parallel to the baseline from the alignment point given by the textAlign attribute to the left side of the bounding rectangle of the given text, in CSS pixels; positive numbers indicating a distance going left from the given alignment point.

Note
The sum of this value and the next (actualBoundingBoxRight) can be wider than the width of the inline box, in particular with slanted fonts where characters overhang their advance width.

actualBoundingBoxRight attribute
The distance parallel to the baseline from the alignment point given by the textAlign attribute to the right side of the bounding rectangle of the given text, in CSS pixels; positive numbers indicating a distance going right from the given alignment point.

fontBoundingBoxAscent attribute
The distance from the horizontal line indicated by the textBaseline attribute to the ascent metric of the first available font, in CSS pixels; positive numbers indicating a distance going up from the given baseline.

Note
This value and the next are useful when rendering a background that have to have a consistent height even if the exact text being rendered changes. The actualBoundingBoxAscent attribute (and its corresponding attribute for the descent) are useful when drawing a bounding box around specific text.

fontBoundingBoxDescent attribute
The distance from the horizontal line indicated by the textBaseline attribute to the descent metric of the first available font, in CSS pixels; positive numbers indicating a distance going down from the given baseline.

actualBoundingBoxAscent attribute
The distance from the horizontal line indicated by the textBaseline attribute to the top of the bounding rectangle of the given text, in CSS pixels; positive numbers indicating a distance going up from the given baseline.

Note
This number can vary greatly based on the input text, even if the first font specified covers all the characters in the input. For example, the actualBoundingBoxAscent of a lowercase "o" from an alphabetic baseline would be less than that of an uppercase "F". The value can easily be negative; for example, the distance from the top of the em box (textBaseline value "top") to the top of the bounding rectangle when the given text is just a single comma "", would likely (unless the font is quite unusual) be negative.

actualBoundingBoxDescent attribute
The distance from the horizontal line indicated by the textBaseline attribute to the bottom of the bounding rectangle of the given text, in CSS pixels; positive numbers indicating a distance going down from the given baseline.

emHeightAscent attribute
The distance from the horizontal line indicated by the textBaseline attribute to the highest top of the em squares in the inline box, in CSS pixels; positive numbers indicating that the given baseline is below the top of that em square (so this value will usually be positive). Zero if the given baseline is the top of that em square; half the font size if the given baseline is the middle of that em square.
**emHeightDescent attribute**

The distance from the horizontal line indicated by the `textBaseline` attribute to the lowest bottom of the em squares in the inline box, in CSS pixels; positive numbers indicating that the given baseline is above the bottom of that em square. (Zero if the given baseline is the bottom of that em square.)

**hangingBaseline attribute**

The distance from the horizontal line indicated by the `textBaseline` attribute to the hanging baseline of the inline box, in CSS pixels; positive numbers indicating that the given baseline is below the hanging baseline. (Zero if the given baseline is the hanging baseline.)

**alphabeticBaseline attribute**

The distance from the horizontal line indicated by the `textBaseline` attribute to the alphabetic baseline of the inline box, in CSS pixels; positive numbers indicating that the given baseline is below the alphabetic baseline. (Zero if the given baseline is the alphabetic baseline.)

**ideographicBaseline attribute**

The distance from the horizontal line indicated by the `textBaseline` attribute to the ideographic-under baseline of the inline box, in CSS pixels; positive numbers indicating that the given baseline is below the ideographic-under baseline. (Zero if the given baseline is the ideographic-under baseline.)

**Note**

Glyphs rendered using `fillText()` and `strokeText()` can spill out of the box given by the font size (the em square size) and the width returned by `measureText()` (the text width). Authors are encouraged to use the bounding box values described above if this is an issue.

**Note**

A future version of the 2D context API might provide a way to render fragments of documents, rendered using CSS, straight to the canvas. This would be provided in preference to a dedicated way of doing multiline layout.

### 4.12.5.1.12 Drawing paths to the canvas

Objects that implement the `CanvasDrawPath` interface have a current default path. There is only one current default path, it is not part of the drawing state. The current default path is a path, as described above.

**For web developers (non-normative)**

```javascript
context.beginPath();
context.fill({ fillRule });
context.fill(path [, fillRule ]);  // Fills the subpaths of the current default path or the given path with the current fill style, obeying the given fill rule.
context.stroke();
context.stroke(path);  // Strokes the subpaths of the current default path or the given path with the current stroke style.
context.clip();
context.clip(path [, fillRule ]);  // Further constrains the clipping region to the current default path or the given path, using the given fill rule to determine what points are in the path.
context.isPointInPath(x, y [, fillRule ]);  // Returns true if the given point is in the current default path or the given path, using the given fill rule to determine what points are in the path.
```
context.isPointInStroke(x, y)
context.isPointInStroke(path, x, y)

Returns true if the given point would be in the region covered by the stroke of the current default path or the given path, given the current stroke style.

The beginPath() method, when invoked, must empty the list of subpaths in the context’s current default path so that it once again has zero subpaths.

Where the following method definitions use the term intended path, it means the Path2D argument, if one was provided, or the current default path otherwise.

When the intended path is a Path2D object, the coordinates and lines of its subpaths must be transformed according to the current transformation matrix on the object implementing the CanvasTransform interface when used by these methods (without affecting the Path2D object itself). When the intended path is the current default path, it is not affected by the transform. (This is because transformations already affect the current default path when it is constructed, so applying it when it is painted as well would result in a double transformation.)

The fill() method, when invoked, must fill all the subpaths of the intended path, using fillStyle, and using the fill rule indicated by the fillRule argument. Open subpaths must be implicitly closed when being filled (without affecting the actual subpaths).

The stroke() method, when invoked, must trace the intended path, using this CanvasPathDrawingStyles object for the line styles, and then fill the resulting path using the strokeStyle attribute, using the nonzero winding rule.

Notes

As a result of how the algorithm to trace a path is defined, overlapping parts of the paths in one stroke operation are treated as if their union was what was painted.

The stroke style is affected by the transformation during painting, even if the intended path is the current default path.

Paths, when filled or stroked, must be painted without affecting the current default path or any Path2D objects, and must be subject to shadow effects, global alpha, the clipping region, and global composition operators. (The effect of transformations is described above and varies based on which path is being used.)

The clip() method, when invoked, must create a new clipping region by calculating the intersection of the current clipping region and the area described by the intended path, using the fill rule indicated by the fillRule argument. Open subpaths must be implicitly closed when computing the clipping region, without affecting the actual subpaths. The new clipping region replaces the current clipping region.

When the context is initialized, the clipping region must be set to the largest infinite surface (i.e. by default, no clipping occurs).

The isPointInPath() method, when invoked, must return true if the point given by the x and y coordinates passed to the method, when treated as coordinates in the canvas coordinate space unaffected by the current transformation, is inside the intended path as determined by the fill rule indicated by the fillRule argument; and must return false otherwise. Open subpaths must be implicitly closed when computing the area inside the path, without affecting the actual subpaths. Points on the path itself must be considered to be inside the path. If either of the arguments are infinite or NaN, then the method must return false.

The isPointInStroke() method, when invoked, must return true if the point given by the x and y coordinates passed to the method, when treated as coordinates in the canvas coordinate space unaffected by the current transformation, is inside the path that results from tracing the intended path, using the nonzero winding rule, and using the CanvasPathDrawingStyles interface for the line styles; and must return false otherwise. Points on the resulting path must be considered to be inside the path. If either of the arguments are infinite or NaN, then the method must return false.
This canvas element has a couple of checkboxes. The path-related commands are highlighted:

```html
<canvas height=400 width=750>
  <label><input type=checkbox id=showA> Show A</label>
  <label><input type=checkbox id=showB> Show Bs</label>
</canvas>

<script>
  function drawCheckbox(context, element, x, y, paint) {
    context.save();
    context.font = '10px sans-serif';
    context.textAlign = 'left';
    context.textBaseline = 'middle';
    var metrics = context.measureText(element.labels[0].textContent);
    if (paint) {
      context.beginPath();
      context.strokeStyle = 'black';
      context.strokeStyle = 'black';
      context.stroke();
      if (element.checked) {
        context.fillStyle = 'black';
        context.fill();
      }
      context.fillText(element.labels[0].textContent, x+5, y);
    }
    context.beginPath();
    context.rect(x, y, 12 + metrics.width, 2, 14);
    context.drawFocusIfNeeded(element);
    context.restore();
  }
  function drawBase() {
  }
  function drawAs() {
  }
  function drawBs() {
  }
  function redraw() {
    var canvas = document.getElementsByTagName('canvas')[0];
    var context = canvas.getContext('2d');
    context.clearRect(0, 0, canvas.width, canvas.height);
    drawCheckbox(context, document.getElementById('showA'), 20, 40, true);
    drawCheckbox(context, document.getElementById('showB'), 20, 60, true);
    drawBase();
    if (document.getElementById('showA').checked) {
      drawAs();
    }
    if (document.getElementById('showB').checked) {
      drawBs();
    }
  }
  function processClick(event) {
    var canvas = document.getElementsByTagName('canvas')[0];
    var context = canvas.getContext('2d');
    var x = event.clientX;
    var y = event.clientY;
    var node = event.target;
    while (node) {
      x -= node.offsetLeft - node.scrollLeft;
      y -= node.offsetTop - node.scrollTop;
      node = node.offsetParent;
    }
    drawCheckbox(context, document.getElementById('showA'), 20, 40, false);
    if (context.isPointInPath(x, y)) {
      document.getElementById('showA').checked = !document.getElementById('showA').checked;
      drawCheckbox(context, document.getElementById('showB'), 20, 60, false);
    }
</script>
```
4.12.5.1.13 Drawing focus rings and scrolling paths into view

For web developers (non-normative)

```javascript
if (context.isPointInPath(x, y))
    document.getElementById('showB').checked = !(document.getElementById('showB').checked);
    redraw();
}
document.getElementsByTagName('canvas')[0].addEventListener('focus', redraw, true);
document.getElementsByTagName('canvas')[0].addEventListener('blur', redraw, true);
document.getElementsByTagName('canvas')[0].addEventListener('change', redraw, true);
document.getElementsByTagName('canvas')[0].addEventListener('click', processClick, false);
redraw();
</script>
```

Objects that implement the CanvasUserInterface interface provide the following methods to control drawing focus rings and scrolling paths into view.

The `drawFocusIfNeeded(element)` method, when invoked, must run these steps:

1. If `element` is not focused or is not a descendant of the element with whose context the method is associated, then return.

2. Draw a focus ring of the appropriate style along the intended path, following platform conventions.

   ```javascript
   context.drawFocusIfNeeded(element)
   ```

   ```javascript
   context.drawFocusIfNeeded(path, element)
   ```

   If the given element is focused, draws a focus ring around the current default path or the given path, following the platform conventions for focus rings.

   ```javascript
   context.drawFocusIfNeeded(path, element)
   ```

   Scrolls the current default path or the given path into view. This is especially useful on devices with small screens, where the whole canvas might not be visible at once.

   ```javascript
   context.scrollPathIntoView(path)
   ```

   Scrolls the current default path or the given path into view.

The `drawFocusIfNeeded(element)` method, when invoked, must run these steps:

1. If `element` is not focused or is not a descendant of the element with whose context the method is associated, then return.

2. Draw a focus ring of the appropriate style along the intended path, following platform conventions.

   ```javascript
   context.drawFocusIfNeeded(element)
   ```

   ```javascript
   context.drawFocusIfNeeded(path, element)
   ```

   If the given element is focused, draws a focus ring around the current default path or the given path, following the platform conventions for focus rings.

   ```javascript
   context.drawFocusIfNeeded(path, element)
   ```

   Scrolls the current default path or the given path into view. This is especially useful on devices with small screens, where the whole canvas might not be visible at once.

   ```javascript
   context.scrollPathIntoView(path)
   ```

   Scrolls the current default path or the given path into view.

For web developers (non-normative)

Some platforms only draw focus rings around elements that have been focused from the keyboard, and not those focused from the mouse. Other platforms simply don’t draw focus rings around some elements at all unless relevant accessibility features are enabled. This API is intended to follow these conventions. User agents that implement distinctions based on the manner in which the element was focused are encouraged to classify focus driven by the `focus()` method based on the kind of user interaction event from which the call was triggered (if any).

The focus ring should not be subject to the shadow effects, the global alpha, the global composition operators, or any of the members in the CanvasFillStrokeStyles, CanvasPathDrawingStyles, CanvasTextDrawingStyles interfaces, but should be subject to the clipping region. (The effect of transformations is described above and varies based on which path is being used.)

3. Inform the user that the focus is at the location given by the intended path. User agents may wait until the next time the event loop reaches its update the rendering step to optionally inform the user.

User agents should not implicitly close open subpaths in the intended path when drawing the focus ring.

This might be a moot point, however. For example, if the focus ring is drawn as an axis-aligned bounding rectangle around the points in the intended path, then whether the subpaths are closed or not has no effect. This specification intentionally does not specify precisely how focus rings are to be drawn: user agents are expected to honor their platform's native conventions.
The `scrollPathIntoView()` method, when invoked, must run these steps:

1. Let `specifiedRectangle` be the rectangle of the bounding box of the intended path.
2. Let `notionalChild` be a hypothetical element that is a rendered child of the `canvas` element whose dimensions are those of `specifiedRectangle`.
3. Scroll `notionalChild` into view with `behavior` set to "auto", `block` set to "start", and `inline` set to "nearest".
4. Optionally, inform the user that the caret or selection (or both) cover `specifiedRectangle` of the canvas. The user agent may wait until the next time the `event loop` reaches its `update the rendering` step to optionally inform the user.

"Inform the user", as used in this section, does not imply any persistent state change. It could mean, for instance, calling a system accessibility API to notify assistive technologies such as magnification tools so that the user’s magnifier moves to the given area of the canvas. However, it does not associate the path with the element, or provide a region for tactile feedback, etc.

### 4.12.5.1.14 Drawing images

Objects that implement the `CanvasDrawImage` interface have the `drawImage` method to draw images.

This method can be invoked with three different sets of arguments:

- `drawImage(image, dx, dy)`
- `drawImage(image, dx, dy, dw, dh)`
- `drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)`

For web developers (non-normative)

```
context . drawImage(image, dx, dy)
context . drawImage(image, dx, dy, dw, dh)
context . drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)
```

Draws the given image onto the canvas. The arguments are interpreted as follows:

- If the image isn’t yet fully decoded, then nothing is drawn. If the image is a canvas with no data, throws an "`InvalidStateError`" `DOMException`.

When the `drawImage()` method is invoked, the user agent must run these steps:

1. If any of the arguments are infinite or NaN, then return.
2. Let \textit{usability} be the result of \texttt{checking the usability of image}.\footnote{670}

3. If \textit{usability} is \textit{bad}, then return (without drawing anything).

4. Establish the source and destination rectangles as follows:

   If not specified, the \texttt{dw} and \texttt{dh} arguments must default to the values of \texttt{sw} and \texttt{sh}, interpreted such that one CSS pixel in the image is treated as one unit in the output bitmap's coordinate space. If the \texttt{sx}, \texttt{sy}, \texttt{sw}, and \texttt{sh} arguments are omitted, then they must default to 0, 0, the image's intrinsic width in image pixels, and the image's intrinsic height in image pixels, respectively. If the image has no intrinsic dimensions, then the concrete object size must be used instead, as determined using the CSS "Concrete Object Size Resolution" algorithm, with the specified size having neither a definite width nor height, nor any additional constraints, the object's intrinsic properties being those of the image argument, and the default object size being the size of the output bitmap.\footnote{643} \cite{CSSIMAGES}

The source rectangle is the rectangle whose corners are the four points (sx, sy), (sx+sw, sy), (sx+sw, sy+sh), (sx, sy+sh).

The destination rectangle is the rectangle whose corners are the four points (dx, dy), (dx+dw, dy), (dx+dw, dy+dh), (dx, dy+dh).

When the source rectangle is outside the source image, the source rectangle must be clipped to the source image and the destination rectangle must be clipped in the same proportion.

\begin{note}
When the destination rectangle is outside the destination image (the output bitmap\footnote{643}), the pixels that land outside the output bitmap\footnote{643} are discarded, as if the destination was an infinite canvas whose rendering was clipped to the dimensions of the output bitmap\footnote{643}.
\end{note}

5. If one of the \texttt{sw} or \texttt{sh} arguments is zero, then return. Nothing is painted.

6. Paint the region of the \texttt{image} argument specified by the source rectangle on the region of the rendering context's output bitmap\footnote{643} specified by the destination rectangle, after applying the current transformation matrix\footnote{659} to the destination rectangle.

The image data must be processed in the original direction, even if the dimensions given are negative.

When scaling up, if the \texttt{imageSmoothingEnabled} attribute is set to true, the user agent should attempt to apply a smoothing algorithm to the image data when it is scaled. User agents which support multiple filtering algorithms may use the value of the \texttt{imageSmoothingQuality} attribute to guide the choice of filtering algorithm when the \texttt{imageSmoothingEnabled} attribute is set to true. Otherwise, the image must be rendered using nearest-neighbor interpolation.

\begin{note}
This specification does not define the precise algorithm to use when scaling an image down, or when scaling an image up when the \texttt{imageSmoothingEnabled} attribute is set to true.
\end{note}

\begin{note}
When a \texttt{canvas} element is drawn onto itself, the \texttt{drawing model} requires the source to be copied before the image is drawn, so it is possible to copy parts of a \texttt{canvas} element onto overlapping parts of itself.
\end{note}

If the original image data is a bitmap image, then the value painted at a point in the destination rectangle is computed by filtering the original image data. The user agent may use any filtering algorithm (for example bilinear interpolation or nearest-neighbor). When the filtering algorithm requires a pixel value from outside the original image data, it must instead use the value from the nearest edge pixel. (That is, the filter uses 'clamp-to-edge' behavior.) When the filtering algorithm requires a pixel value from outside the source rectangle but inside the original image data, then the value from the original image data must be used.

\begin{note}
Thus, scaling an image in parts or in whole will have the same effect. This does mean that when sprites coming from a single sprite sheet are to be scaled, adjacent images in the sprite sheet can interfere. This can be avoided by ensuring each sprite in the sheet is surrounded by a border of transparent black, or by copying sprites to be scaled into temporary \texttt{canvas} elements and drawing the scaled sprites from there.
\end{note}

Images are painted without affecting the current path, and are subject to \texttt{shadow effects}, \texttt{global alpha}, the \texttt{clipping region}, and \texttt{global composition operators}.\footnote{679}
7. If image is not origin-clean, then set the CanvasRenderingContext2D's origin-clean flag to false.

### 4.12.5.1.15 Pixel manipulation

**For web developers (non-normative)**

```javascript
imagedata = new ImageData(sw, sh)
```

Returns an `ImageData` object with the given dimensions. All the pixels in the returned object are transparent black.

Throws an `IndexSizeError` DOMException if either of the width or height arguments are zero.

```javascript
imagedata = context.createImageData(sw, sh)
```

Returns an `ImageData` object with the same dimensions as the argument. All the pixels in the returned object are transparent black.

```javascript
imagedata = context.createImageData(imagedata)
```

Returns an `ImageData` object using the data provided in the `Uint8ClampedArray` argument, interpreted using the given dimensions.

As each pixel in the data is represented by four numbers, the length of the data needs to be a multiple of four times the given width. If the height is provided as well, then the length needs to be exactly the width times the height times 4.

Throws an `IndexSizeError` DOMException if the given data and dimensions can't be interpreted consistently, or if either dimension is zero.

```javascript
imagedata = context.getImageData(sx, sy, sw, sh)
```

Returns an `ImageData` object containing the image data for the given rectangle of the bitmap.

Throws an `IndexSizeError` DOMException if either of the width or height arguments are zero.

```javascript
imagedata.width
```

Returns the actual dimensions of the data in the `ImageData` object, in pixels.

```javascript
imagedata.height
```

Returns the one-dimensional array containing the data in RGBA order, as integers in the range 0 to 255.

```javascript
context.putImageData(imagedata, dx, dy [, dirtyX, dirtyY, dirtyWidth, dirtyHeight ])
```

Paints the data from the given `ImageData` object onto the bitmap. If a dirty rectangle is provided, only the pixels from that rectangle are painted.

The `globalAlpha` and `globalCompositeOperation` attributes, as well as the shadow attributes, are ignored for the purposes of this method call; pixels in the canvas are replaced wholesale, with no composition, alpha blending, no shadows, etc.

Throws an `InvalidStateError` DOMException if the `imagedata` object's `data` attribute value's `[[ViewedArrayBuffer]]` internal slot is detached.

Objects that implement the `CanvasImageData` interface provide the following methods for reading and writing pixel data to the bitmap.

The `ImageData()` constructors and the `createImageData()` methods are used to instantiate new `ImageData` objects.

When the `ImageData()` constructor is invoked with two numeric arguments `sw` and `sh`, it must run these steps:

1. If one or both of `sw` and `sh` are zero, then throw an `IndexSizeError` DOMException.
2. Create an `ImageData` object with parameter `pixelsPerRow` set to `sw`, and `rows` set to `sh`.
3. Initialize the image data of the newly created `ImageData` object to transparent black.
4. Return the newly created `ImageData` object.

When the `ImageData()` constructor is invoked with its first argument being an `Uint8ClampedArray` `source` and its second and optional third arguments being numeric arguments `sw` and `sh`, it must run these steps:

1. Let `length` be the number of bytes in `source`.
2. If `length` is not a nonzero integral multiple of four, then throw an "InvalidStateError" DOMException.

3. Let `length` be `length` divided by four.

4. If `length` is not an integral multiple of `sw`, then throw an "IndexSizeError" DOMException.

   Note
   At this step, the length is guaranteed to be greater than zero (otherwise the second step above would have aborted the steps), so if `sw` is zero, this step will throw the exception and return.

5. Let `height` be `length` divided by `sw`.

6. If the `sh` argument was not omitted, and its value is not equal to `height`, then throw an "IndexSizeError" DOMException.

7. Create an ImageData object, with parameter `pixelsPerRow` set to `sw`, `rows` set to `sh`, and using `source`. Return the newly created ImageData object.

   Note
   The resulting object's data is not a copy of source, it's the actual Uint8ClampedArray object passed as the first argument to the constructor.

When the `createImageData()` method is invoked with two numeric arguments `sw` and `sh`, it must create an ImageData object, with parameter `pixelsPerRow` set to the absolute magnitude of `sw`, and parameter `rows` set to the absolute magnitude of `sh`. Initialize the image data of the new ImageData object to transparent black. If both `sw` and `sh` are nonzero, then return the new ImageData object. If one or both of `sw` and `sh` are zero, then throw an "IndexSizeError" DOMException instead.

When the `createImageData()` method is invoked with a single `imagedata` argument, it must create an ImageData object, with parameter `pixelsPerRow` set to the value of the `width` attribute of the ImageData object, object passed as the argument, and the `rows` parameter set to the value of the `height` attribute. Initialize the image data of the new ImageData object to transparent black. Return the newly created ImageData object.

The `getImageData()` method, when invoked, must, if either the `sw` or `sh` arguments are zero, throw an "InvalidStateError" DOMException; otherwise, if the CanvasRenderingContext2D's `origin-clean` flag is set to false, it must throw an "SecurityError" DOMException; otherwise, it must create an ImageData object, with parameter `pixelsPerRow` set to `sw`, and parameter `rows` set to `sh`. Set the pixel values of the image data of the newly created ImageData object to represent the output bitmap for the area of that bitmap denoted by the rectangle whose corners are the four points (`sx`, `sy`), (`sx+sw`, `sy`), (`sx+sw`, `sy+sh`), (`sx`, `sy+sh`), in the bitmap's coordinate space units. Pixels outside the output bitmap must be set to transparent black. Pixel values must not be premultiplied by alpha.

When the user agent is required to create an ImageData object, given a positive integer number of rows `rows`, a positive integer number of pixels per row `pixelsPerRow`, and an optional Uint8ClampedArray source, it must run these steps:

1. Let `imageData` be a new uninitialized ImageData object.

2. If `source` is specified, then assign the `data` attribute of `imageData` to `source`.

3. If `source` is not specified, then initialize the `data` attribute of `imageData` to a new Uint8ClampedArray object. The Uint8ClampedArray object must use a new Canvas Pixel ArrayBuffer for its storage, and must have a zero start offset and a length equal to the length of its storage, in bytes. The Canvas Pixel ArrayBuffer must have the correct size to store `rows x pixelsPerRow` pixels.

   If the Canvas Pixel ArrayBuffer cannot be allocated, then rethrow the Range Error thrown by JavaScript, and return.

4. Initialize the `width` attribute of `imageData` to `pixelsPerRow`.

5. Initialize the `height` attribute of `imageData` to `rows`.

6. Return `imageData`.

ImageData objects are serializable objects. Their serialization steps, given value and serialized, are:

1. Set serialized.[[Data]] to the sub-serialization of the value's `data` attribute.

2. Set serialized.[[Width]] to the value of value's `width` attribute.
3. Set `serialized.[[Height]]` to the value of value's `height` attribute.

Their deserialization steps, given `serialized` and `value`, are:

1. Initialize value's `data` attribute to the sub-deserialization of `serialized.[[Data]]`.
2. Initialize value's `width` attribute to `serialized.[[Width]]`.
3. Initialize value's `height` attribute to `serialized.[[Height]]`.

A Canvas Pixel `ArrayBuffer` is an `ArrayBuffer` whose data is represented in left-to-right order, row by row top to bottom, starting with the top left, with each pixel's red, green, blue, and alpha components being given in that order for each pixel. Each component of each pixel represented in this array must be in the range 0..255, representing the 8 bit value for that component. The components must be assigned consecutive indices starting with 0 for the top left pixel's red component.

The `putImageData()` method writes data from `ImageData` structures back to the rendering context's `output bitmap`. Its arguments are: `imagedata, dx, dy, dirtyX, dirtyY, dirtyWidth, and dirtyHeight`.

When the last four arguments to this method are omitted, they must be assumed to have the values 0, 0, the `width` member of the `imagedata` structure, and the `height` member of the `imagedata` structure, respectively.

The method, when invoked, must act as follows:

1. Let `buffer` be `imagedata`'s `data` attribute value's `[[ViewedArrayBuffer]]` internal slot.
2. If `isDetachedBuffer(buffer)` is true, then throw an "InvalidStateError" DOMException.
3. If `dirtyWidth` is negative, then let `dirtyX` be `dirtyX+dirtyWidth`, and let `dirtyWidth` be equal to the absolute magnitude of `dirtyWidth`.
   
   If `dirtyHeight` is negative, then let `dirtyY` be `dirtyY+dirtyHeight`, and let `dirtyHeight` be equal to the absolute magnitude of `dirtyHeight`.
4. If `dirtyX` is negative, then let `dirtyWidth` be `dirtyWidth+dirtyX`, and let `dirtyX` be zero.
   
   If `dirtyY` is negative, then let `dirtyHeight` be `dirtyHeight+dirtyY`, and let `dirtyY` be zero.
5. If `dirtyX+dirtyWidth` is greater than the `width` attribute of the `imagedata` argument, then let `dirtyWidth` be the value of that `width` attribute, minus the value of `dirtyX`.
   
   If `dirtyY+dirtyHeight` is greater than the `height` attribute of the `imagedata` argument, then let `dirtyHeight` be the value of that `height` attribute, minus the value of `dirtyY`.
6. If, after those changes, either `dirtyWidth` or `dirtyHeight` are negative or zero, then return without affecting any bitmaps.
7. For all integer values of `x` and `y` where `dirtyX ≤ x < dirtyX+dirtyWidth` and `dirtyY ≤ y < dirtyY+dirtyHeight`, copy the four channels of the pixel with coordinate `(x, y)` in the `imagedata` data structure's `Canvas Pixel ArrayBuffer` to the pixel with coordinate `(dx+x, dy+y)` in the rendering context's `output bitmap`.

Note

Due to the lossy nature of converting to and from premultiplied alpha color values, pixels that have just been set using `putImageData()` might be returned to an equivalent `getImageData` as different values.

The current path, transformation matrix, shadow attributes, global alpha, the clipping region, and global composition operator must not affect the methods described in this section.

Example

In the following example, the script generates an `ImageData` object so that it can draw onto it.

```javascript
// canvas is a reference to a <canvas> element
var context = canvas.getContext('2d');

// create a blank slate
var data = context.createImageData(canvas.width, canvas.height);
```
// create some plasma
FillPlasma(data, 'green'); // green plasma

// add a cloud to the plasma
AddCloud(data, data.width/2, data.height/2); // put a cloud in the middle

// paint the plasma+cloud on the canvas
context.putImageData(data, 0, 0);

// support methods
function FillPlasma(data, color) {
    ...
}
function AddCloud(data, x, y) {
    ...
}

Here is an example of using `getImageData()` and `putImageData()` to implement an edge detection filter.

```html
<!DOCTYPE HTML>
<html lang="en">
<head>
    <title>Edge detection demo</title>
    <script>
        var image = new Image();
        function init() {
            image.onload = demo;
            image.src = "image.jpeg";
        }
        function demo() {
            var canvas = document.getElementsByTagName('canvas')[0];
            var context = canvas.getContext('2d');

            // draw the image onto the canvas
            context.drawImage(image, 0, 0);

            // get the image data to manipulate
            var input = context.getImageData(0, 0, canvas.width, canvas.height);

            // get an empty slate to put the data into
            var output = context.createImageData(canvas.width, canvas.height);

            // alias some variables for convenience
            // In this case input.width and input.height
            // match canvas.width and canvas.height
            // but we'll use the former to keep the code generic.
            var w = input.width, h = input.height;
            var inputData = input.data;
            var outputData = output.data;

            // edge detection
            for (var y = 1; y < h-1; y += 1) {
                for (var x = 1; x < w-1; x += 1) {
                    for (var c = 0; c < 3; c += 1) {
                        var i = (y*w + x)*4 + c;
                    }
                }
            }
        }
    </script>
</head>
<body>
</body>
</html>
```
All drawing operations on an object which implements the CanvasCompositing interface are affected by the global compositing attributes, `globalAlpha` and `globalCompositeOperation`.

The `globalAlpha` attribute gives an alpha value that is applied to shapes and images before they are composited onto the output bitmap. The value must be in the range from 0.0 (fully transparent) to 1.0 (no additional transparency). If an attempt is made to set the attribute to a value outside this range, including Infinity and Not-a-Number (NaN) values, then the attribute must retain its previous value. When the context is created, the `globalAlpha` attribute must initially have the value 1.0.

The `globalCompositeOperation` attribute sets the current composition operator, which controls how shapes and images are drawn onto the output bitmap, once they have had `globalAlpha` and the current transformation matrix applied. The possible values are those defined in Compositing and Blending. These values are all case-sensitive — they must be used exactly as defined. User agents must not recognize values that are not identical to one of the values given in Compositing and Blending.

On setting, if the user agent does not recognize the specified value, it must be ignored, leaving the value of `globalCompositeOperation` unaffected. Otherwise, the attribute must be set to the given new value.

When the context is created, the `globalCompositeOperation` attribute must initially have the value `source-over`.

### 4.12.5.1.17 Image smoothing

For web developers (non-normative)

The `imageSmoothingEnabled` attribute returns whether pattern fills and the `drawImage()` method will attempt to smooth images if their pixels don't line up exactly with the display, when scaling images up. Can be set, to change whether images are smoothed (true) or not (false).
Objects that implement the `CanvasImageSmoothing` interface have attributes that control how image smoothing is performed.

The `imageSmoothingEnabled` attribute, on getting, must return the last value it was set to. On setting, it must be set to the new value. When the object implementing the `CanvasImageSmoothing` interface is created, the attribute must be set to true.

The `imageSmoothingQuality` attribute, on getting, must return the last value it was set to. On setting, it must be set to the new value. When the object implementing the `CanvasImageSmoothing` interface is created, the attribute must be set to "low".

4.12.5.18 Shadows

All drawing operations on an object which implements the `CanvasShadowStyles` interface are affected by the four global shadow attributes.

For web developers (non-normative)

- `context.shadowColor`
  - Returns the current shadow color.
  - Can be set, to change the shadow color. Values that cannot be parsed as CSS colors are ignored.

- `context.shadowOffsetX` and `context.shadowOffsetY`
  - Returns the current shadow offset.
  - Can be set, to change the shadow offset. Values that are not finite numbers are ignored.

- `context.shadowBlur` (non-normative)
  - Returns the current level of blur applied to shadows.
  - Can be set, to change the blur level. Values that are not finite numbers greater than or equal to zero are ignored.

The `shadowColor` attribute sets the color of the shadow.

When the context is created, the `shadowColor` attribute initially must be `transparent black`.

On getting, the `serialization of the color` must be returned.

On setting, the new value must be `parsed` with this `canvas` element and the color assigned. If parsing the value results in failure then it must be ignored, and the attribute must retain its previous value. (`CSSCOLOR`)

The `shadowOffsetX` and `shadowOffsetY` attributes specify the distance that the shadow will be offset in the positive horizontal and positive vertical distance respectively. Their values are in coordinate space units. They are not affected by the current transformation matrix.

When the context is created, the shadow offset attributes must initially have the value 0.

On getting, they must return their current value. On setting, the attribute being set must be set to the new value, except if the value is infinite or NaN, in which case the new value must be ignored.

The `shadowBlur` attribute specifies the level of the blurring effect. (The units do not map to coordinate space units, and are not affected by the current transformation matrix.)

When the context is created, the `shadowBlur` attribute must initially have the value 0.

On getting, the attribute must return its current value. On setting the attribute must be set to the new value, except if the value is negative, infinite or NaN, in which case the new value must be ignored.
Shadows are only drawn if the opacity component of the alpha component of the color of `shadowColor` is nonzero and either the `shadowBlur` is nonzero, or the `shadowOffsetX` is nonzero, or the `shadowOffsetY` is nonzero.

When shadows are drawn, they must be rendered as follows:

1. Let A be an infinite transparent black bitmap on which the source image for which a shadow is being created has been rendered.
2. Let B be an infinite transparent black bitmap, with a coordinate space and an origin identical to A.
3. Copy the alpha channel of A to B, offset by `shadowOffsetX` in the positive x direction, and `shadowOffsetY` in the positive y direction.
4. If `shadowBlur` is greater than 0:
   1. Let σ be half the value of `shadowBlur`.
   2. Perform a 2D Gaussian Blur on B, using σ as the standard deviation.
5. Set the red, green, and blue components of every pixel in B to the red, green, and blue components (respectively) of the color of `shadowColor`.
6. Multiply the alpha component of every pixel in B by the alpha component of the color of `shadowColor`.
7. The shadow is in the bitmap B, and is rendered as part of the drawing model described below.

If the current composition operation is `copy`, then shadows effectively won't render (since the shape will overwrite the shadow).

### 4.12.5.1.19 Filters

All drawing operations on an object which implements the `CanvasFilters` interface are affected by the global `filter` attribute.

<table>
<thead>
<tr>
<th>For web developers (non-normative)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>context.filter [ = value ]</code></td>
</tr>
<tr>
<td>Returns the current filter.</td>
</tr>
<tr>
<td>Can be set, to change the filter.</td>
</tr>
<tr>
<td>Values that cannot be parsed as a</td>
</tr>
<tr>
<td><code>&lt;filter-value-list&gt;</code> value are ignored.</td>
</tr>
</tbody>
</table>

The `filter` attribute, on getting, must return the last value it was successfully set to. The value must not be re-serialized. On setting, if the new value is 'none' (not the empty string, null, or undefined), filters must be disabled for the context. Otherwise, the value must be parsed as a `<filter-value-list>` value. If the value cannot be parsed as a `<filter-value-list>` value, where using property-independent style sheet syntax like 'inherit' or 'initial' is considered an invalid value, then it must be ignored, and the attribute must retain its previous value. When creating the object implementing the `CanvasFilters` interface, the attribute must be set to 'none'.

A `<filter-value-list>` value consists of a sequence of one or more filter functions or references to SVG filters. The input to the filter is used as the input to the first item in the list. Subsequent items take the output of the previous item as their input. [FILTERS]

Coordinates used in the value of the `filter` attribute are interpreted such that one pixel is equivalent to one SVG user space unit and to one canvas coordinate space unit. Filter coordinates are not affected by the `current transformation matrix`. The current transformation matrix affects only the input to the filter. Filters are applied in the `output bitmap`'s coordinate space.

When the value of the `filter` attribute defines lengths using percentages or using 'em' or 'ex' units, these must be interpreted relative to the `computed value` of the 'font-size' property of the `font style source object` at the time that the attribute is set, if it is an element. If the `computed values` are undefined for a particular case (e.g. because the `font style source object` is not an element or is not being rendered), then the relative keywords must be interpreted relative to the default value of the `font` attribute. The ‘larger’ and ‘smaller’ keywords are not supported.

If the value of the `filter` attribute refers to an SVG filter in the same document, and this SVG filter changes, then the changed filter is used for the next draw operation.

If the value of the `filter` attribute refers to an SVG filter in an external resource document and that document is not loaded when
a drawing operation is invoked, then the drawing operation must proceed with no filtering.

### 4.12.5.1.20 Working with externally-defined SVG filters

*This section is non-normative.*

Since drawing is performed using filter value 'none' until an externally-defined filter has finished loading, authors might wish to determine whether such a filter has finished loading before proceeding with a drawing operation. One way to accomplish this is to load the externally-defined filter elsewhere within the same page in some element that sends a *load* event (for example, an SVG *use* element), and wait for the *load* event to be dispatched.

### 4.12.5.1.21 Drawing model

When a shape or image is painted, user agents must follow these steps, in the order given (or act as if they do):

1. Render the shape or image onto an infinite transparent black bitmap, creating image $A$, as described in the previous sections. For shapes, the current fill, stroke, and line styles must be honored, and the stroke must itself also be subjected to the current transformation matrix.

2. When the filter attribute is set to a value other than 'none' and all the externally-defined filters it references, if any, are in documents that are currently loaded, then use image $A$ as the input to the filter, creating image $B$. Otherwise, let $B$ be an alias for $A$.

3. *When shadows are drawn*, render the shadow from image $B$, using the current shadow styles, creating image $C$.

4. *When shadows are drawn*, multiply the alpha component of every pixel in $C$ by $globalAlpha$.

5. *When shadows are drawn*, composite $C$ within the clipping region over the current output bitmap using the current composition operator.

6. Multiply the alpha component of every pixel in $B$ by $globalAlpha$.

7. Composite $B$ within the clipping region over the current output bitmap using the current composition operator.

When compositing onto the output bitmap, pixels that would fall outside of the output bitmap must be discarded.

### 4.12.5.1.22 Best practices

When a canvas is interactive, authors should include *focusable* elements in the element's fallback content corresponding to each focusable part of the canvas, as in the example above.

When rendering focus rings, to ensure that focus rings have the appearance of native focus rings, authors should use the `drawFocusIfNeeded()` method, passing it the element for which a ring is being drawn. This method only draws the focus ring if the element is focused, so that it can simply be called whenever drawing the element, without checking whether the element is focused or not first.

In addition to drawing focus rings, authors should use the `scrollPathIntoView()` method when an element in the canvas is focused, to make sure it is visible on the screen (if applicable).

Authors should avoid implementing text editing controls using the *canvas* element. Doing so has a large number of disadvantages:

- Mouse placement of the caret has to be reimplemented.
- Keyboard movement of the caret has to be reimplemented (possibly across lines, for multiline text input).
- Scrolling of the text control has to be implemented (horizontally for long lines, vertically for multiline input).
- Native features such as copy-and-paste have to be reimplemented.
- Native features such as spell-checking have to be reimplemented.
- Native features such as drag-and-drop have to be reimplemented.
Native features such as page-wide text search have to be reimplemented.

Native features specific to the user, for example custom text services, have to be reimplemented. This is close to impossible since each user might have different services installed, and there is an unbounded set of possible such services.

Bidirectional text editing has to be reimplemented.

For multiline text editing, line wrapping has to be implemented for all relevant languages.

Text selection has to be reimplemented.

Dragging of bidirectional text selections has to be reimplemented.

Platform-native keyboard shortcuts have to be reimplemented.

Platform-native input method editors (IMEs) have to be reimplemented.

Undo and redo functionality has to be reimplemented.

Accessibility features such as magnification following the caret or selection have to be reimplemented.

This is a huge amount of work, and authors are most strongly encouraged to avoid doing any of it by instead using the input element, the textarea element, or the contenteditable attribute.

This section is non-normative.

### 4.12.5.1.23 Examples

This section is non-normative.

**Example**

Here is an example of a script that uses canvas to draw pretty glowing lines.

```html
<canvas width="800" height="450"></canvas>

<script>

var context = document.getElementsByTagName('canvas')[0].getContext('2d');

var lastX = context.canvas.width * Math.random();
var lastY = context.canvas.height * Math.random();
var hue = 0;
function line() {
  context.save();
  context.translate(context.canvas.width/2, context.canvas.height/2);
  context.scale(0.9, 0.9);
  context.translate(-context.canvas.width/2, -context.canvas.height/2);
  context.beginPath();
  context.lineWidth = 5 + Math.random() * 10;
  context.moveTo(lastX, lastY);
  lastX = context.canvas.width * Math.random();
  lastY = context.canvas.height * Math.random();
  context.bezierCurveTo(context.canvas.width * Math.random(),
                        context.canvas.height * Math.random(),
                        context.canvas.width * Math.random(),
                        context.canvas.height * Math.random(),
                        lastX, lastY);

  hue = hue + 10 * Math.random();
  context.strokeStyle = 'hsl(' + hue + ', 50%, 50%)';
  context.shadowColor = 'white';
  context.shadowBlur = 10;
  context.stroke();
  context.restore();
}
setInterval(line, 50);
</script>
```
function blank() {
    context.fillStyle = 'rgba(0,0,0,0.1)';
    context.fillRect(0, 0, context.canvas.width, context.canvas.height);
}
setInterval(blank, 40);

The 2D rendering context for canvas is often used for sprite-based games. The following example demonstrates this:

Blue Robot Player Sprite by JohnColburn. Licensed under the terms of the Creative Commons Attribution Share-Alike 3.0 Unported license. This work is itself licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
context.fillRect(0, 0, width, this.horizon);
context.fillStyle = this.earth;
context.fillRect(0, this.horizon, width, height-this.horizon);
// then, draw the cloudy banner
// we make it cloudy by having the draw text off the top of the
// canvas, and just having the blurred shadow shown on the canvas
context.save();
context.translate(width-((this.offset+(this.width*3.2)) % (this.width*4.0))+0, 0);
context.shadowColor = 'white';
context.shadowOffsetY = 30-this.horizon/3; // offset down on canvas
context.shadowBlur = '5';
context.fillStyle = 'white';
context.textAlign = 'left';
context.textBaseline = 'top';
context.font = '20px sans-serif';
context.fillText('WHATWG ROCKS', 10, -30); // text up above canvas
context.restore();

// then, draw the background tree
context.save();
context.translate(width-((this.offset+(this.width*0.2)) % (this.width*1.5))+30, 0);
context.beginPath();
context.fillStyle = 'rgb(143,89,2)';
context.strokeStyle = 'rgb(10,10,10)';
context.lineWidth = 2;
context.rect(0, this.horizon+5, 10, -50); // trunk
context.fill();
context.stroke();
context.beginPath();
context.fillStyle = 'rgb(78,154,6)';
context.strokeStyle = 'rgb(220,154,94)';
context.lineWidth = 2;
context.arc(5, this.horizon-60, 30, 0, Math.PI*2); // leaves
context.fill();
context.stroke();
context.restore();

this.paintForeground = function (context, width, height) {
// draw the box that goes in front
context.save();
context.translate(width-((this.offset+(this.width*0.7)) % (this.width*1.1))+0, 0);
context.beginPath();
context.rect(0, this.horizon - 5, 25, 25);
context.fillStyle = 'rgb(220,154,94)';
context.strokeStyle = 'rgb(10,10,10)';
context.lineWidth = 2;
context.fill();
context.stroke();
context.restore();

};
}
</script>
<script>
var BlueRobot = function () {
this.sprites = new Image();
this.sprites.src = 'blue-robot.png'; // this sprite sheet has 8 cells
this.targetMode = 'idle';
this.walk = function () {
    this.targetMode = 'walk';
};
this.stop = function () {
    this.targetMode = 'idle';
};
</script>
this.frameIndex = {
    'idle': [0], // first cell is the idle frame
    'walk': [1,2,3,4,5,6], // the walking animation is cells 1-6
    'stop': [7], // last cell is the stopping animation
};
this.mode = 'idle';
this.frame = 0; // index into frameIndex
this.tick = function () {
    // this advances the frame and the robot
    // the return value is how many pixels the robot has moved
    this.frame += 1;
    if (this.frame >= this.frameIndex[this.mode].length) {
        // we've reached the end of this animation cycle
        this.frame = 0;
        if (this.mode != this.targetMode) {
            // switch to next cycle
            if (this.mode == 'walk') {
                // we need to stop walking before we decide what to do next
                this.mode = 'stop';
            } else if (this.mode == 'stop') {
                if (this.targetMode == 'walk')
                    this.mode = 'walk';
                else
                    this.mode = 'idle';
            } else if (this.mode == 'idle') {
                if (this.targetMode == 'walk')
                    this.mode = 'walk';
            } else
                this.mode = 'idle';
        }
    }
}
if (this.mode == 'walk')
    return 8;
return 0;
this.paint = function (context, x, y) {
    if (!this.sprites.complete) return;
    // draw the right frame out of the sprite sheet onto the canvas
    // we assume each frame is as high as the sprite sheet
    // the x,y coordinates give the position of the bottom center of the sprite
    context.drawImage(this.sprites,
        this.frameIndex[this.mode][this.frame] * this.sprites.height, 0,
        this.sprites.height, this.sprites.height,
        x - this.sprites.height/2, y - this.sprites.height, this.sprites.height,
        this.sprites.height);
};
</script>
<script>
var canvas = document.getElementsByTagName('canvas')[0];
var context = canvas.getContext('2d');
var landscape = new Landscape(context, canvas.width, canvas.height);
var blueRobot = new BlueRobot();
// paint when the browser wants us to, using requestAnimationFrame()
function paint() {
    context.clearRect(0, 0, canvas.width, canvas.height);
    landscape.paintBackground(context, canvas.width, canvas.height);
    blueRobot.paintBackground(context, canvas.width/2, landscape.horizon*1.1);
    landscape.paintForeground(context, canvas.width, canvas.height);
    requestAnimationFrame(paint);
}
4.12.5.2 The ImageBitmap rendering context

4.12.5.2.1 Introduction

ImageBitmapRenderingContext is a performance-oriented interface that provides a low overhead method for displaying the contents of ImageBitmap objects. It uses transfer semantics to reduce overall memory consumption. It also streamlines performance by avoiding intermediate compositing, unlike the drawImage() method of CanvasRenderingContext2D.

Using an img element as an intermediate for getting an image resource into a canvas, for example, would result in two copies of the decoded image existing in memory at the same time: the img element's copy, and the one in the canvas's backing store. This memory cost can be prohibitive when dealing with extremely large images. This can be avoided by using ImageBitmapRenderingContext.

Example

Using ImageBitmapRenderingContext, here is how to transcode an image to the JPEG format in a memory- and CPU-efficient way:

```javascript
createImageBitmap(inputImageBlob).then(image => {
  const canvas = document.createElement('canvas');
  const context = canvas.getContext('bitmaprenderer');
  context.transferFromImageBitmap(image);

  canvas.toBlob(outputJPEGBlob => {
    // Do something with outputJPEGBlob.
  }, 'image/jpeg');
});
```

4.12.5.2.2 The ImageBitmapRenderingContext interface

```idl
[Exposed=(Window,Worker)]
interface ImageBitmapRenderingContext {
  readonly attribute (HTMLCanvasElement or OffscreenCanvas) canvas;
  undefined transferFromImageBitmap(ImageBitmap? bitmap);
};
```

dictionary ImageBitmapRenderingContextSettings {
boolean alpha = true;
;
For web developers (non-normative)

canvas = context.canvas('bitmaprender', [ [alpha: false] ])

Returns an ImageBitmapRenderingContext object that is permanently bound to a particular canvas element. If the alpha setting is provided and set to false, then the canvas is forced to always be opaque.

canvas = context.canvas()

Returns the canvas element that the context is bound to.

canvas = context.transferFromImageBitmap(imageBitmap)

Transfers the underlying bitmap data from imageBitmap to context, and the bitmap becomes the contents of the canvas element to which context is bound.

canvas = context.transferFromImageBitmap(null)

Replaces contents of the canvas element to which context is bound with a transparent black bitmap whose size corresponds to the width and height content attributes of the canvas element.

The canvas attribute must return the value it was initialized to when the object was created.

An ImageBitmapRenderingContext object has an output bitmap, which is a reference to bitmap data.

An ImageBitmapRenderingContext object has a bitmap mode, which can be set to valid or blank. A value of valid indicates that the context's output bitmap refers to bitmap data that was acquired via transferFromImageBitmap(). A value blank indicates that the context's output bitmap is a default transparent bitmap.

An ImageBitmapRenderingContext object also has an alpha flag, which can be set to true or false. When an ImageBitmapRenderingContext object has its alpha flag set to false, the contents of the canvas element to which the context is bound are obtained by compositing the context's output bitmap onto an opaque black bitmap of the same size using the source-over composite operation. If the alpha flag is set to true, then the output bitmap is used as the contents of the canvas element to which the context is bound.

Note

The step of compositing over an opaque black bitmap ought to be elided whenever equivalent results can be obtained more efficiently by other means.

When a user agent is required to set an ImageBitmapRenderingContext's output bitmap, with a context argument that is an ImageBitmapRenderingContext object and an optional argument bitmap that refers to bitmap data, it must run these steps:

1. If a bitmap argument was not provided, then:
   1. Set context's bitmap mode to blank.
   2. Let canvas be the canvas element to which context is bound.
   3. Set context's output bitmap to be transparent black with an intrinsic width equal to the numeric value of canvas's width attribute and an intrinsic height equal to the numeric value of canvas's height attribute, those values being interpreted in CSS pixels.
   4. Set the output bitmap's origin-clean flag to true.

2. If a bitmap argument was provided, then:
   1. Set context's bitmap mode to valid.
   2. Set context's output bitmap to refer to the same underlying bitmap data as bitmap, without making a copy.

Note

The origin-clean flag of bitmap is included in the bitmap data to be referenced by context's output bitmap.
The **ImageBitmapRenderingContext** creation algorithm, which is passed a `target` and `options`, consists of running these steps:

1. Let `settings` be the result of converting `options` to the dictionary type `ImageBitmapRenderingContextSettings`\(^{687}\). (This can throw an exception.)
2. Let `context` be a new `ImageBitmapRenderingContext`\(^{687}\) object.
3. Initialize `context`'s `canvas`\(^{644}\) attribute to point to `target`.
4. Set `context`'s `output bitmap`\(^{688}\) to the same bitmap as `target`'s bitmap (so that they are shared).
5. Run the steps to set an `ImageBitmapRenderingContext`'s `output bitmap`\(^{688}\) with `context`.
6. Initialize `context`'s `alpha`\(^{688}\) flag to true.
7. Process each of the members of `settings` as follows:
   - `alpha` If false, then set `context`'s `alpha`\(^{688}\) flag to false.
8. Return `context`.

The `transferFromImageBitmap(imageBitmap)` method, when invoked, must run these steps:

1. Let `bitmapContext` be the `ImageBitmapRenderingContext`\(^{687}\) object on which the `transferFromImageBitmap()`\(^{689}\) method was called.
2. If `imageBitmap` is null, then run the steps to set an `ImageBitmapRenderingContext`'s `output bitmap`\(^{688}\), with `bitmapContext` as the `context` argument and no `bitmap` argument, then return.
3. If the value of `imageBitmap`'s `[[Detached]]`\(^{104}\) internal slot is set to true, then throw an "**InvalidStateError**" DOMException.
4. Run the steps to set an `ImageBitmapRenderingContext`'s `output bitmap`\(^{688}\), with the `context` argument equal to `bitmapContext`, and the `bitmap` argument referring to `imageBitmap`'s underlying `bitmap data`\(^{974}\).
5. Set the value of `imageBitmap`'s `[[Detached]]`\(^{104}\) internal slot to true.
6. Unset `imageBitmap`'s `bitmap data`\(^{974}\).

4.12.5.3 The **OffscreenCanvas**\(^{689}\) interface \(^{568}\)

```idl
typedef (OffscreenCanvasRenderingContext2D or ImageBitmapRenderingContext or WebGLRenderingContext or WebGL2RenderingContext) OffscreenRenderingContext;

dictionary ImageEncodeOptions {
    DOMString type = "image/png";
    unrestricted double quality;
};

enum OffscreenRenderingContextId { "2d", "bitmaprenderer", "webgl", "webgl2" };

[Exposed=(Window,Worker), Transferable]
interface OffscreenCanvas : EventTarget {
    constructor([EnforceRange] unsigned long long width, [EnforceRange] unsigned long long height);

    attribute [EnforceRange] unsigned long long width;
    attribute [EnforceRange] unsigned long long height;

    OffscreenRenderingContext? getContext(OffscreenRenderingContextId contextId, optional any options = null);
    ImageBitmap transferToImageBitmap();
    Promise<Blob> convertToBlob(optional ImageEncodeOptions options = {});
};
```
OffscreenCanvas objects are used to create rendering contexts, much like an HTMLCanvasElement, but with no connection to the DOM. This makes it possible to use canvas rendering contexts in workers.

An OffscreenCanvas object may hold a weak reference to a placeholder canvas element, which is typically in the DOM, whose embedded content is provided by the OffscreenCanvas object. The bitmap of the OffscreenCanvas object is pushed to the placeholder canvas element by calling the commit() method of the OffscreenCanvas object's rendering context. All rendering context types that can be created by an OffscreenCanvas object must implement a commit() method. The exact behavior of the commit method (e.g. whether it copies or transfers bitmaps) may vary, as defined by the rendering contexts' respective specifications. Only the 2D context for offscreen canvases is defined in this specification.

For web developers (non-normative)

offscreenCanvas = new OffscreenCanvas(width, height)

Returns a new OffscreenCanvas object that is not linked to a placeholder canvas element, and whose bitmap's size is determined by the width and height arguments.

context = offscreenCanvas.getContext(contextId [, options ])

Returns an object that exposes an API for drawing on the OffscreenCanvas object. contextId specifies the desired API: "2d", "bitmaprenderer", "webgl", or "webgl2". options is handled by that API.

This specification defines the "2d" context below, which is similar but distinct from the "2d" context that is created from a canvas element. The WebGL specifications define the "webgl" and "webgl2" contexts. [WEBGL1]

Returns null if the canvas has already been initialized with another context type (e.g., trying to get a "2d" context after getting a "webgl" context).

An OffscreenCanvas object has an internal bitmap that is initialized when the object is created. The width and height of the bitmap are equal to the values of the width and height attributes of the OffscreenCanvas object. Initially, all the bitmap's pixels are transparent black.

An OffscreenCanvas object 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, an OffscreenCanvas object also has a context mode, which is initially none but can be changed to either 2d, bitmaprenderer, webgl, webgl2, or detached by algorithms defined in this specification.

The constructor OffscreenCanvas(width, height), when invoked, must create a new OffscreenCanvas object with its bitmap initialized to a rectangular array of transparent black pixels of the dimensions specified by width and height; and its width and height attributes initialized to width and height respectively.

OffscreenCanvas objects are transferable. Their transfer steps, given value and dataHolder, are as follows:

1. If value's context mode is not equal to none, then throw an "InvalidStateError" DOMException.
2. Set value's context mode to detached.
3. Let width and height be the dimensions of value's bitmap.
4. Unset value's bitmap.
5. Set dataHolder.[[Width]] to width and dataHolder.[[Height]] to height.
6. Set dataHolder.[[PlaceholderCanvas]] to be a weak reference to value's placeholder canvas element, if value has one, or null if it does not.

Their transfer-receiving steps, given dataHolder and value, are:

1. Initialize value's bitmap to a rectangular array of transparent black pixels with width given by dataHolder.[[Width]] and height given by dataHolder.[[Height]].
2. If dataHolder.[[PlaceholderCanvas]] is not null, set value's placeholder canvas element to dataHolder.[[PlaceholderCanvas]] (while maintaining the weak reference semantics).
The `getContext(contextId, options)` method of an `OffscreenCanvas` object, when invoked, must run these steps:

1. If `options` is not an `object`, then set `options` to null.
2. Set `options` to the result of converting `options` to a JavaScript value.
3. Run the steps in the cell of the following table whose column header matches this `bitmap` and whose row header matches `contextId`:

<table>
<thead>
<tr>
<th>&quot;2d&quot;</th>
<th>bitmaprenderer</th>
<th>&quot;webgl2&quot;</th>
<th>detached</th>
</tr>
</thead>
<tbody>
<tr>
<td>Follow the <code>offscreen 2D context creation algorithm</code> defined in the section below, passing it this <code>OffscreenCanvas</code> object and options, to obtain an <code>OffscreenCanvasRenderingContext2D</code> object; if this does not throw an exception, then set this <code>OffscreenCanvas</code> object's <code>context mode</code> to <code>2d</code>, and return the new <code>OffscreenCanvasRenderingContext2D</code> object.</td>
<td>Return the same object as was returned the last time the method was invoked with this same first argument.</td>
<td>Return null.</td>
<td>Return null.</td>
</tr>
<tr>
<td>Follow the <code>ImageBitmapRenderingContext creation algorithm</code> defined in the section above, passing it this <code>OffscreenCanvas</code> object and options, to obtain an <code>ImageBitmapRenderingcontext</code> object; if this does not throw an exception, then set this <code>OffscreenCanvas</code> object's <code>context mode</code> to <code>bitmaprenderer</code>, and return the new <code>ImageBitmapRenderingcontext</code> object.</td>
<td>Return null.</td>
<td>Return null.</td>
<td>Return null.</td>
</tr>
<tr>
<td>Follow the instructions given in the WebGL specifications’ Context Creation sections to obtain either a <code>WebGLRenderingContext</code>, <code>WebGL2RenderingContext</code>, or null; if the returned value is null, then return null; otherwise, set this <code>OffscreenCanvas</code> object's <code>context mode</code> to <code>webgl2</code> or <code>webgl</code>, and return the <code>WebGLRenderingContext</code> or <code>WebGL2RenderingContext</code> object.</td>
<td>Return null.</td>
<td>Return null.</td>
<td>Return the same value as was returned the last time the method was invoked with this same first argument.</td>
</tr>
</tbody>
</table>

For web developers (non-normative)

```
offscreenCanvas.width [ = value ]
offscreenCanvas.height [ = value ]
```

These attributes return the dimensions of the `OffscreenCanvas` object's `bitmap`.

They can be set, to replace the `bitmap` with a new, `transparent black` bitmap of the specified dimensions (effectively resizing it).

If either the `width` or `height` attributes of an `OffscreenCanvas` object are set (to a new value or to the same value as before) and the `OffscreenCanvas` object's `context mode` is `2d`, then replace the `OffscreenCanvas` object's `bitmap` with a new `transparent black` bitmap and reset the rendering context to its default state. The new bitmap's dimensions are equal to the new values of the `width` and `height` attributes.

The resizing behavior for "**webgl2"" and "**webgl2" contexts is defined in the WebGL specifications.

Note

If an `OffscreenCanvas` object whose dimensions were changed has a `placeholder canvas element`'s `intrinsic size` will only be updated via the `commit()` method of the `OffscreenCanvas` object's rendering context.

For web developers (non-normative)

```
promise = offscreenCanvas.convertToBlob([options])
```

Returns a promise that will fulfill with a new `Blob` object representing a file containing the image in the `OffscreenCanvas` object.
The argument, if provided, is a dictionary that controls the encoding options of the image file to be created. The `type` field specifies the file format and has a default value of "image/png", that type is also used if the requested type isn't supported. If the image format supports variable quality (such as "image/jpeg"), then the `quality` field is a number in the range 0.0 to 1.0 inclusive indicating the desired quality level for the resulting image.

**canvas.transferToImageBitmap()**

Returns a newly created `ImageBitmap` object with the image in the `OffscreenCanvas` object. The image in the `OffscreenCanvas` object is replaced with a new blank image.

The `convertToBlob(options)` method, when invoked, must run the following steps:

1. If the value of this `OffscreenCanvas` object's `[[Detached]]` internal slot is set to true, then return a promise rejected with an "InvalidStateError" DOMException.
2. If this `OffscreenCanvas` object's `context mode` is `2d` and the rendering context's `bitmap`'s `origin-clean` flag is set to false, then return a promise rejected with a "SecurityError" DOMException.
3. If this `OffscreenCanvas` object's `bitmap` has no pixels (i.e., either its horizontal dimension or its vertical dimension is zero) then return a promise rejected with an "IndexSizeError" DOMException.
4. Let `bitmap` be a copy of this `OffscreenCanvas` object's `bitmap`.
5. Let `result` be a new promise object.
6. Run these steps in parallel:
   1. Let `file` be a serialization of `bitmap` as a file, with options's `type` and `quality` if present.
   2. Queue an element task on the canvas blob serialization task source given the `canvas` element to run these steps:
      1. If `file` is null, then reject `result` with an "EncodingError" DOMException.
      2. Otherwise, resolve `result` with a new `Blob` object, created in the relevant Realm of this `OffscreenCanvas` object, representing `file`.
    
7. Return `result`.

The `transferToImageBitmap()` method, when invoked, must run the following steps:

1. If the value of this `OffscreenCanvas` object's `[[Detached]]` internal slot is set to true, then throw an "InvalidStateError" DOMException.
2. If this `OffscreenCanvas` object's `context mode` is `none`, then throw an "InvalidStateError" DOMException.
3. Let `image` be a newly created `ImageBitmap` object that references the same underlying bitmap data as this `OffscreenCanvas` object's `bitmap`.
4. Set this `OffscreenCanvas` object's `bitmap` to reference a newly created bitmap of the same dimensions as the previous bitmap, and with its pixels initialized to transparent black, or opaque black if the rendering context's `alpha` flag is set to false.

**Note**

This means that if the rendering context of this `OffscreenCanvas` is a WebGLRenderingContext, the value of `preserveDrawingBuffer` will have no effect.

5. Return `image`.

### 4.12.5.3.1 The offscreen 2D rendering context

```idl
[Exposed=[Window,Worker]]
interface OffscreenCanvasRenderingContext2D {
undefined commit();
```
The OffscreenCanvasRenderingContext2D object is a rendering context for drawing to the bitmap of an OffscreenCanvas object. It is similar to the CanvasRenderingContext2D object, with the following differences:

- there is no support for user interface features;
- its canvas attribute refers to an OffscreenCanvas object rather than a canvas element;
- it has a commit() method for pushing the rendered image to the context's OffscreenCanvas object's placeholder canvas element.

An OffscreenCanvasRenderingContext2D object has a bitmap that is initialized when the object is created.

The bitmap has an origin-clean flag, which can be set to true or false. Initially, when one of these bitmaps is created, its origin-clean flag must be set to true.

An OffscreenCanvasRenderingContext2D object also has an alpha flag, which can be set to true or false. Initially, when the context is created, its alpha flag must be set to true. When an OffscreenCanvasRenderingContext2D object has its alpha flag set to false, then its alpha channel must be fixed to 1.0 (fully opaque) for all pixels, and attempts to change the alpha component of any pixel must be silently ignored.

An OffscreenCanvasRenderingContext2D object has an associated OffscreenCanvas object, which is the OffscreenCanvas object from which the OffscreenCanvasRenderingContext2D object was created.

For web developers (non-normative)

offscreenCanvasRenderingContext2D . commit()

Copies the rendering context's bitmap to the bitmap of the placeholder canvas element of the associated OffscreenCanvas object. The copy operation is synchronous. Calling this method is not needed for the transfer, since it happens automatically during the event loop execution.

offscreenCanvas = offscreenCanvasRenderingContext2D . canvas

Returns the associated OffscreenCanvas object.

The offscreen 2D context creation algorithm, which is passed a target (an OffscreenCanvas object) and optionally some arguments, consists of running the following steps:

1. If the algorithm was passed some arguments, let arg be the first such argument. Otherwise, let arg be undefined.
2. Let settings be the result of converting options to the dictionary type CanvasRenderingContext2DSettings. (This can throw an exception.).
3. Let context be a new OffscreenCanvasRenderingContext2D object.
4. Set context's associated OffscreenCanvas object to target.
5. Process each of the members of settings as follows:

   **alpha**

   If false, set context’s **alpha** flag to false.

6. Set context’s **bitmap** to a newly created bitmap with the dimensions specified by the **width** and **height** attributes of target, and set target’s bitmap to the same bitmap (so that they are shared).

7. If context’s **alpha** flag is set to true, initialize all the pixels of context’s **bitmap** to transparent black. Otherwise, initialize the pixels to opaque black.

8. Return context.

The **commit()** method, when invoked, must run the following steps:

1. If this **OffscreenCanvasRenderingContext2D**’s associated **OffscreenCanvas object** does not have a **placeholder canvas element**, then return.

2. Let image be a copy of this **OffscreenCanvasRenderingContext2D**’s **bitmap**, including the value of its **origin-clean** flag.

3. Queue an element task on the DOM manipulation task source given the **placeholder canvas element**’s **output bitmap** to be a reference to image.

   **Note**

   If image has different dimensions than the bitmap previously referenced as the **placeholder canvas element**’s **output bitmap**, then this task will result in a change in the **placeholder canvas element**’s intrinsic size, which can affect document layout.

   **Note**

   Implementations are encouraged to short-circuit the graphics update steps of the window event loop, for the purposes of updating the contents of a **placeholder canvas element** to the display. This could mean, for example, that the **commit()** method can copy the bitmap contents directly to a graphics buffer that is mapped to the physical display location of the **placeholder canvas element**. This or similar short-circuiting approaches can significantly reduce display latency, especially in cases where the **commit()** method is invoked from a worker event loop and the window event loop of the **placeholder canvas element** is busy. However, such shortcuts can not have any script-observable side-effects. This means that the committed bitmap still needs to be sent to the **placeholder canvas element**, in case the element is used as a **CanvasImageSource**, as an **ImageBitmapSource**, or in case **toDataURL()** or **toBlob()** are called on it.

The **canvas** attribute, on getting, must return this **OffscreenCanvasRenderingContext2D**’s associated **OffscreenCanvas object**.

### 4.12.5.4 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 **HTML5SVGImageElement** 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, when invoked, 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.

4.12.5.5 Serializing bitmaps to a file

When a user agent is to create a serialization of the bitmap as a file, given a type and an optional quality, it must create an image file in the format given by type. If an error occurs during the creation of the image file (e.g. an internal encoder error), then the result of the serialization is null. [PNG]

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 type is supplied, then it must be interpreted as a MIME type giving the format to use. If the type has any parameters, then it must be treated as not supported.

Example

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, then 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 an opaque black background using the source-over operator.

If type is an image format that supports variable quality (such as "image/jpeg"), quality is given, and type is not "image/png", then, if type is Number, and quality is in the range 0.0 to 1.0 inclusive, the user agent must treat quality as the desired quality level. Otherwise, the user agent must use its default quality value, as if the quality argument had not been given.

Note

The use of type-testing here, instead of simply declaring quality as a Web IDL double, is a historical artifact.

Note

Different implementations can have slightly different interpretations of "quality". When the quality is not specified, an implementation-specific default is used that represents a reasonable compromise between compression ratio, image quality, and encoding time.

4.12.5.6 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 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 changing the value of the `width` or the `height` content attribute of the canvas element to which a CanvasRenderingContext2D is bound, the bitmap is cleared and its `origin-clean` flag is reset.

When using an ImageBitmapRenderingContext, the value of the `origin-clean` flag is propagated from ImageBitmap objects when they are transferred to the canvas via `transferFromImageBitmap()`.

### 4.13 Custom elements

#### 4.13.1 Introduction

This section is non-normative.

Custom elements provide a way for authors to build their own fully-featured DOM elements. Although authors could always use non-standard elements in their documents, with application-specific behavior added after the fact by scripting or similar, such elements have historically been non-conforming and not very functional. By defining a custom element, authors can inform the parser how to properly construct an element and how elements of that class should react to changes.

Custom elements are part of a larger effort to "rationalise the platform", by explaining existing platform features (like the elements of HTML) in terms of lower-level author-exposed extensibility points (like custom element definition). Although today there are many limitations on the capabilities of custom elements—both functionally and semantically—that prevent them from fully explaining the behaviors of HTML's existing elements, we hope to shrink this gap over time.

#### 4.13.1.1 Creating an autonomous custom element

This section is non-normative.

For the purposes of illustrating how to create an autonomous custom element, let's define a custom element that encapsulates rendering a small icon for a country flag. Our goal is to be able to use it like so:

```html
<flag-icon country="nl"></flag-icon>
```

To do this, we first declare a class for the custom element, extending HTMLElement:

```javascript
class FlagIcon extends HTMLElement {
  constructor() {
    super();
    this._countryCode = null;
  }

  static get observedAttributes() { return "country"; }

  attributeChangedCallback(name, oldValue, newValue) {
    // name will always be "country" due to observedAttributes
    this._countryCode = newValue;
    this._updateRendering();
  }

  connectedCallback() {
    this._updateRendering();
  }

  get country() {
```

696
```javascript
return this._countryCode;
}
set country(v) {
    this.setAttribute("country", v);
}

_updateRendering() {
    // Left as an exercise for the reader. But, you'll probably want to
    // check this.ownerDocument.defaultView to see if we've been
    // inserted into a document with a browsing context, and avoid
    // doing any work if not.
}
}
```

We then need to use this class to define the element:

```javascript
customElements.define("flag-icon", FlagIcon);
```

At this point, our above code will work! The parser, whenever it sees the `flag-icon` tag, will construct a new instance of our `FlagIcon` class, and tell our code about its new `country` attribute, which we then use to set the element's internal state and update its rendering (when appropriate).

You can also create `flag-icon` elements using the DOM API:

```javascript
const flagIcon = document.createElement("flag-icon")
flagIcon.country = "jp"
document.body.appendChild(flagIcon)
```

Finally, we can also use the custom element constructor itself. That is, the above code is equivalent to:

```javascript
const flagIcon = new FlagIcon()
flagIcon.country = "jp"
document.body.appendChild(flagIcon)
```

### 4.13.1.2 Creating a form-associated custom element

This section is non-normative.

Adding a static `formAssociated` property, with a true value, makes an autonomous custom element a form-associated custom element. The `ElementInternals` interface helps you to implement functions and properties common to form control elements.

```javascript
class MyCheckbox extends HTMLElement {
    static get formAssociated() { return true; }
    static get observedAttributes() { return ['checked']; }

    constructor() {
        super();
        this._internals = this.attachInternals();
        this.addEventListener('click', this._onClick.bind(this));
    }

    get form() { return this._internals.form; }
    get name() { return this.getAttribute('name'); }
    get type() { return this.localName; }

    get checked() { return this.getAttribute('checked'); }
    set checked(flag) { this.toggleAttribute('checked', Boolean(flag)); }

    attributeChangedCallback(name, oldValue, newValue) {
```
You can use the custom element `my-checkbox` like a built-in form-associated element. For example, putting it in `form` or `label` associates the `my-checkbox` element with them, and submitting the `form` will send data provided by `my-checkbox` implementation.

```html
<form action="..." method="...">
  <label><my-checkbox name="agreed"></my-checkbox> I read the agreement.</label>
  <input type="submit">
</form>
```

### 4.13.1.3 Creating a custom element with default accessible roles, states, and properties

This section is non-normative.

By using the appropriate properties of `ElementInternals`, your custom element can have default accessibility semantics. The following code expands our form-associated checkbox from the previous section to properly set its default role and checkedness, as viewed by accessibility technology:

```javascript
class MyCheckbox extends HTMLElement {
  static get formAssociated() { return true; }
  static get observedAttributes() { return ['checked']; }

  constructor() {
    super();
    this._internals = this.attachInternals();
    this.addEventListener('click', this._onClick.bind(this));

    this._internals.role = 'checkbox';
    this._internals.ariaChecked = false;
  }

  get form() { return this._internals.form; }
  get name() { return this.getAttribute('name'); }
  get type() { return this.localName; }

  get checked() { return this.getAttribute('checked'); }
  set checked(flag) { this.toggleAttribute('checked', Boolean(flag)); }

  attributeChangedCallback(name, oldValue, newValue) {
    // name will always be "checked" due to observedAttributes
    this._internals.setFormValue(this.checked ? 'on' : null);
    this._internals.ariaChecked = this.checked;
  }

  _onClick(event) {
    this.checked = !this.checked;
  }
}
customElements.define('my-checkbox', MyCheckbox);
```

Note that, like for built-in elements, these are only defaults, and can be overridden by the page author using the `role` and `aria-*` attributes:
Custom element authors are encouraged to state what aspects of their accessibility semantics are strong native semantics, i.e., should not be overridden by users of the custom element. In our example, the author of the my-checkbox element would state that its role and aria-checked values are strong native semantics, thus discouraging code such as the above.

4.13.1.4 Creating a customized built-in element

This section is non-normative.

Customized built-in elements are a distinct kind of custom element, which are defined slightly differently and used very differently compared to autonomous custom elements. They exist to allow reuse of behaviors from the existing elements of HTML, by extending those elements with new custom functionality. This is important since many of the existing behaviors of HTML elements can unfortunately not be duplicated by using purely autonomous custom elements. Instead, customized built-in elements allow the installation of custom construction behavior, lifecycle hooks, and prototype chain onto existing elements, essentially "mixing in" these capabilities on top of the already-existing element.

Customized built-in elements require a distinct syntax from autonomous custom elements because user agents and other software key off an element's local name in order to identify the element's semantics and behavior. That is, the concept of customized built-in elements building on top of existing behavior depends crucially on the extended elements retaining their original local name.

In this example, we'll be creating a customized built-in element named plastic-button, which behaves like a normal button but gets fancy animation effects added whenever you click on it. We start by defining a class, just like before, although this time we extend HTMLButtonElement instead of HTMLElement:

```javascript
class PlasticButton extends HTMLButtonElement {
  constructor() {
    super();
    
    this.addEventListener("click", () => {
      // Draw some fancy animation effects!
    });
  }
}
```

When defining our custom element, we have to also specify the extends option:

```javascript
customElements.define("plastic-button", PlasticButton, { extends: "button" });
```

In general, the name of the element being extended cannot be determined simply by looking at what element interface it extends, as many elements share the same interface (such as q and blockquote both sharing HTMLQuoteElement).

To construct our customized built-in element from parsed HTML source text, we use the is attribute on a button element:

```html
<button is="plastic-button">Click Me!</button>
```

Trying to use a customized built-in element as an autonomous custom element will not work; that is, `<plastic-button>Click me?</plastic-button>` will simply create an HTMLElement with no special behavior.

If you need to create a customized built-in element programmatically, you can use the following form of createElement:

```javascript
const plasticButton = document.createElement("button", { is: "plastic-button" });
plasticButton.textContent = "Click me!";
```
And as before, the constructor will also work:

```javascript
const plasticButton2 = new PlasticButton();
console.log(plasticButton2.localName); // will output "button"
console.assert(plasticButton2 instanceof PlasticButton);
console.assert(plasticButton2 instanceof HTMLButtonElement);
```

Note that when creating a customized built-in element programmatically, the `is` attribute will not be present in the DOM, since it was not explicitly set. However, it will be added to the output when serializing:

```javascript
console.assert(!plasticButton.hasAttribute("is"));
console.log(plasticButton.outerHTML); // will output '<button is="plastic-button"></button>'
```

Regardless of how it is created, all of the ways in which `button` is special apply to such "plastic buttons" as well: their focus behavior, ability to participate in form submission, the `disabled` attribute, and so on.

**Customized built-in elements** are designed to allow extension of existing HTML elements that have useful user-agent supplied behavior or APIs. As such, they can only extend existing HTML elements defined in this specification, and cannot extend legacy elements such as `bgsound`, `blink`, `isindex`, `keygen`, `multicol`, `nextid`, or `spacer` that have been defined to use `HTMLUnknownElement` as their element interface. One reason for this requirement is future-compatibility: if a customized built-in element was defined that extended a currently-unknown element, for example `combobox`, this would prevent this specification from defining a `combobox` element in the future, as consumers of the derived customized built-in element would have come to depend on their base element having no interesting user-agent-supplied behavior.

### 4.13.1.5 Drawbacks of autonomous custom elements

This section is non-normative.

As specified below, and alluded to above, simply defining and using an element called `taco-button` does not mean that such elements represent buttons. That is, tools such as web browsers, search engines, or accessibility technology will not automatically treat the resulting element as a button just based on its defined name.

To convey the desired button semantics to a variety of users, while still using an autonomous custom element, a number of techniques would need to be employed:

- The addition of the `tabindex` attribute would make the `taco-button` focusable. Note that if the `taco-button` were to become logically disabled, the `tabindex` attribute would need to be removed.
- The addition of an ARIA role and various ARIA states and properties helps convey semantics to accessibility technology. For example, setting the role to "button" will convey the semantics that this is a button, enabling users to successfully interact with the control using usual button-like interactions in their accessibility technology. Setting the `aria-label` property is necessary to give the button an accessible name, instead of having accessibility technology traverse its child text nodes and announce them. And setting the `aria-disabled` state to "true" when the button is logically disabled conveys to accessibility technology the button's disabled state.
- The addition of event handlers to handle commonly-expected button behaviors helps convey the semantics of the button to web browser users. In this case, the most relevant event handler would be one that proxies appropriate `keydown` events to become `click` events, so that you can activate the button both with keyboard and by clicking.
- In addition to any default visual styling provided for `taco-button` elements, the visual styling will also need to be updated to reflect changes in logical state, such as becoming disabled; that is, whatever style sheet has rules for `taco-button` will also need to have rules for `taco-button[disabled]`.

With these points in mind, a full-featured `taco-button` that took on the responsibility of conveying button semantics (including the ability to be disabled) might look something like this:

```javascript
class TacoButton extends HTMLElement {
  static get observedAttributes() { return ["disabled"]; }
}```
constructor() {
    super();
    this._internals = this.attachInternals();
    this._internals.role = "button";

    this.addEventListener("keydown", e => {
        if (e.code === "Enter" || e.code === "Space") {
            this.dispatchEvent(new PointerEvent("click", {
                bubbles: true,
                cancelable: true
            }));
        }
    });

    this.addEventListener("click", e => {
        if (this.disabled) {
            e.preventDefault();
            e.stopImmediatePropagation();
        }
    });

    this._observer = new MutationObserver(() => {
        this._internals.ariaLabel = this.textContent;
    });
}

connectedCallback() {
    this.setAttribute("tabindex", "0");

    this._observer.observe(this, {
        childList: true,
        characterData: true,
        subtree: true
    });

disconnectedCallback() {
    this._observer.disconnect();
}

get disabled() {
    return this.hasAttribute("disabled");
}

set disabled(flag) {
    this.toggleAttribute("disabled", Boolean(flag));
}

attributeChangedCallback(name, oldValue, newValue) {
    // name will always be "disabled" due to observedAttributes
    if (this.disabled) {
        this.removeAttribute("tabindex");
        this._internals.ariaDisabled = "true";
    } else {
        this.setAttribute("tabindex", "0");
        this._internals.ariaDisabled = "false";
    }
}
}

Even with this rather-complicated element definition, the element is not a pleasure to use for consumers: it will be continually "sprouting" tabindex attributes of its own volition, and its choice of tabindex="0" focusability behavior may not match the button behavior on the current platform. This is because as of now there is no way to specify default focus behavior for custom
elements, forcing the use of the `tabindex` attribute to do so (even though it is usually reserved for allowing the consumer to override default behavior).

In contrast, a simple customized built-in element\(^\text{p704}\), as shown in the previous section, would automatically inherit the semantics and behavior of the `button` element, with no need to implement these behaviors manually. In general, for any elements with nontrivial behavior and semantics that build on top of existing elements of HTML, customized built-in elements\(^\text{p704}\) will be easier to develop, maintain, and consume.

4.13.1.6 Upgrading elements after their creation \(^\text{p70}\)

This section is non-normative.

Because element definition\(^\text{p707}\) can occur at any time, a non-custom element could be created, and then later become a custom element\(^\text{p704}\) after an appropriate definition\(^\text{p706}\) is registered. We call this process “upgrading” the element, from a normal element into a custom element.

Upgrades\(^\text{p710}\) enable scenarios where it may be preferable for custom element definitions\(^\text{p706}\) to be registered after relevant elements have been initially created, such as by the parser. They allow progressive enhancement of the content in the custom element. For example, in the following HTML document the element definition for `img-viewer` is loaded asynchronously:

```html
<!DOCTYPE html>
<html lang="en">
<title>Image viewer example</title>

<img-viewer filter="Kelvin">
  <img src="images/tree.jpg" alt="A beautiful tree towering over an empty savannah">
</img-viewer>

<script src="js/elements/img-viewer.js" async></script>
```

The definition for the `img-viewer` element here is loaded using a `script` element marked with the `async` attribute, placed after the `<img-viewer>` tag in the markup. While the script is loading, the `img-viewer` element will be treated as an undefined element, similar to a `span`. Once the script loads, it will define the `img-viewer` element, and the existing `img-viewer` element on the page will be upgraded, applying the custom element's definition (which presumably includes applying an image filter identified by the string "Kelvin", enhancing the image's visual appearance).

Note that upgrades\(^\text{p718}\) only apply to elements in the document tree. (Formally, elements that are connected.) An element that is not inserted into a document will stay un-upgraded. An example illustrates this point:

```html
<!DOCTYPE html>
<html lang="en">
<title>Upgrade edge-cases example</title>

<example-element/>

<script>
"use strict";

const inDocument = document.querySelector("example-element");
const outOfDocument = document.createElement("example-element");

// Before the element definition, both are HTMLElement:
console.assert(inDocument instanceof HTMLElement);
console.assert(outOfDocument instanceof HTMLElement);

class ExampleElement extends HTMLElement {}
customElements.define("example-element", ExampleElement);

// After element definition, the in-document element was upgraded:
```

---

\(^\text{p707}\) element definition
\(^\text{p704}\) customized built-in element
\(^\text{p706}\) definition
\(^\text{p703}\) filter
\(^\text{p702}\) src
\(^\text{p701}\) async
\(^\text{p70}\) upgrade
\(^\text{p700}\) upgrade edge-cases
\(^\text{p70}\) use strict
\(^\text{p613}\) script
\(^\text{p615}\) async
\(^\text{p612}\) instanceof
\(^\text{p535}\) button
\(^\text{p503}\) src
\(^\text{p413}\) script
console.assert(inDocument instanceof ExampleElement);
console.assert(!(outOfDocument instanceof ExampleElement));

document.body.appendChild(outOfDocument);

// Now that we've moved the element into the document, it too was upgraded:
console.assert(outOfDocument instanceof ExampleElement);
</script>

### 4.13.2 Requirements for custom element constructors and reactions

When authoring custom element constructors, authors are bound by the following conformance requirements:

- A parameter-less call to `super()` must be the first statement in the constructor body, to establish the correct prototype chain and `this` value before any further code is run.
- A `return` statement must not appear anywhere inside the constructor body, unless it is a simple early-return (`return` or `return this`).
- The constructor must not use the `document.write()` or `document.open()` methods.
- The element's attributes and children must not be inspected, as in the non-`upgrade` case none will be present, and relying on upgrades makes the element less usable.
- The element must not gain any attributes or children, as this violates the expectations of consumers who use the `createElement` or `createElementNS` methods.
- In general, work should be deferred to `connectedCallback` as much as possible—especially work involving fetching resources or rendering. However, note that `connectedCallback` can be called more than once, so any initialization work that is truly one-time will need a guard to prevent it from running twice.
- In general, the constructor should be used to set up initial state and default values, and to set up event listeners and possibly a shadow root.

Several of these requirements are checked during element creation, either directly or indirectly, and failing to follow them will result in a custom element that cannot be instantiated by the parser or DOM APIs. This is true even if the work is done inside a constructor-initiated microtask, as a microtask checkpoint can occur immediately after construction.

When authoring custom element reactions, authors should avoid manipulating the node tree as this can lead to unexpected results.

#### Example

An element's `connectedCallback` can be queued before the element is disconnected, but as the callback queue is still processed, it results in a `connectedCallback` for an element that is no longer connected:

```javascript
const parent = new CParent(),
child = new CChild();
parent.appendChild(child);
```
A custom element is an element that is custom. Informally, this means that its constructor and prototype are defined by the author, instead of by the user agent. This author-supplied constructor function is called the custom element constructor.

Two distinct types of custom elements can be defined:

1. An autonomous custom element, which is defined with no extends option. These types of custom elements have a local name equal to their defined name.

2. A customized built-in element, which is defined with an extends option. These types of custom elements have a local name equal to the value passed in their extends option, and their defined name is used as the value of the is attribute, which therefore must be a valid custom element name.

After a custom element is created, changing the value of the is attribute does not change the element's behavior, as it is saved on the element as its is value.

Autonomous custom elements have the following element definition:

Categories:
- Flow content
- Phrasing content
- Palpable content

For form-associated custom elements: Listed, labelable, submittable, and ressettable form-associated element.

Contexts in which this element can be used:
- Where phrasing content is expected.

Content model:
- Transparent

Content attributes:
- Global attributes, except the is attribute
  - form, for form-associated custom elements — Associates the element with a form element
  - disabled, for form-associated custom elements — Whether the form control is disabled
  - name, for form-associated custom elements — Name of the element to use for form submission and in the form.elements API

Accessibility considerations:
- For form-associated custom elements: for authors; for implementers.
- Otherwise: for authors; for implementers.

DOM interface:
- Supplied by the element's author (inherits from HTMLElement)

An autonomous custom element does not have any special meaning: it represents its children. A customized built-in element inherits the semantics of the element that it extends.

Any namespace-less attribute that is relevant to the element's functioning, as determined by the element's author, may be specified on an autonomous custom element, so long as the attribute name is XML-compatible and contains no ASCII upper alphas. The exception is the is attribute, which must not be specified on an autonomous custom element (and which will have no effect if it
Customized built-in elements\textsuperscript{5704} follow the normal requirements for attributes, based on the elements they extend. To add custom attribute-based behavior, use \texttt{data-*} attributes.

An autonomous custom element\textsuperscript{704} is called a form-associated custom element if the element is associated with a custom element definition\textsuperscript{706} whose form-associated field is set to true.

The \texttt{name} attribute represents the form-associated custom element’s name. The \texttt{disabled} attribute is used to make the form-associated custom element non-interactive and to prevent its submission value from being submitted. The \texttt{form} attribute is used to explicitly associate the form-associated custom element with its form owner.\texttt{566}

The \texttt{readonly} attribute of form-associated custom elements\textsuperscript{705} specifies that the element is barred from constraint validation\textsuperscript{590}. User agents don’t provide any other behavior for the attribute, but custom element authors should, where possible, use its presence to make their control non-editable in some appropriate fashion, similar to the behavior for the \texttt{readonly} attribute on built-in form controls.

Constraint validation: If the \texttt{readonly} attribute is specified on a form-associated custom element, the element is barred from constraint validation.\textsuperscript{590}

The reset algorithm\textsuperscript{463} for form-associated custom elements is to enqueue a custom element callback reaction with the element, callback name "formResetCallback", and an empty argument list.

A valid custom element name is a sequence of characters name that meets all of the following requirements:

- \texttt{name} must match the PotentialCustomElementName production:

\[
\text{PotentialCustomElementName ::= [a-z] (PCENChar)* '.' (PCENChar)*}
\]

\[
\text{PCENChar ::= 
"." | "." | [0-9] | "," | [a-z] | #x87 | [#xc0-#xd6] | [#xd8-#xf6] | [#xf8-#x37d] | [#x37f-#x1fff] | [#x200c-#x200d] | [#x203f-#x2040] | [#x2070-#x218f] | [#x2c00-#x2fe5] | [#x3001-#x3aff] | [#xf900-#xfdf] | [#xfffd] | [#x10000-#xef]}
\]

This uses the\textsuperscript{EBNF notation} from the XML specification.\textsuperscript{XML}

- \texttt{name} must not be any of the following:
  - annotation-xml
  - color-profile
  - font-face
  - font-face-src
  - font-face- uri
  - font-face-format
  - font-face-name
  - missing-glyph

\textbf{Note}\textsuperscript{705} The list of names above is the summary of all hyphen-containing element names from the applicable specifications, namely SVG 2 and MathML.\textsuperscript{SVG} MathML\textsuperscript{MATHML}

\textbf{Note}\textsuperscript{705} These requirements ensure a number of goals for valid custom element names:\textsuperscript{705}:

- They start with an ASCII lower alpha, ensuring that the HTML parser will treat them as tags instead of as text.
- They do not contain any ASCII upper alphas, ensuring that the user agent can always treat HTML elements ASCII-case-insensitively.
- They contain a hyphen, used for namespacing and to ensure forward compatibility (since no elements will be added to HTML, SVG, or MathML with hyphen-containing local names in the future).
- They can always be created with \texttt{createElement()} and \texttt{createElementNS()}, which have restrictions that go beyond the
A custom element definition describes a custom element and consists of:

A name
A valid custom element name

A local name
A local name

A constructor
A Web IDL CustomElementConstructor callback function type value wrapping the custom element constructor

A list of observed attributes
A sequence<DOMString>

A collection of lifecycle callbacks
A map, whose keys are the strings "connectedCallback", "disconnectedCallback", "adoptedCallback", "attributeChangedCallback", "formAssociatedCallback", "formDisabledCallback", "formResetCallback", and "formStateRestoreCallback". The corresponding values are either a Web IDL Function callback function type value, or null. By default the value of each entry is null.

A construction stack
A list, initially empty, that is manipulated by the upgrade an element algorithm and the HTML element constructors. Each entry in the list will be either an element or an already constructed marker.

A form-associated boolean
If this is true, user agent treats elements associated to this custom element definition as form-associated custom elements.

A disable internals boolean
Controls attachInternals().

A disable shadow boolean
Controls attachShadow().

To look up a custom element definition, given a document, namespace, localName, and is, perform the following steps. They will return either a custom element definition or null:

1. If namespace is not the HTML namespace, return null.
2. If document's browsing context is null, return null.
3. Let registry be document's relevant global object's CustomElementRegistry object.
4. If there is custom element definition in registry with name and local name both equal to localName, return that custom element definition.
5. If there is a custom element definition in registry with name equal to is and local name equal to localName, return that custom element definition.
6. Return null.

4.13.4 The CustomElementRegistry interface

Each Window object is associated with a unique instance of a CustomElementRegistry object, allocated when the Window object is created.

Note
Custom element registries are associated with Window objects, instead of Document objects, since each custom element...
The `customElements` attribute of the `Window` interface must return the `CustomElementRegistry` object for that `Window` object.

```idl
[Exposed=Window]
interface CustomElementRegistry {
  [CEReactions] undefined define(DOMString name, CustomElementConstructor constructor, optional ElementDefinitionOptions options = {});
  (CustomElementConstructor or undefined) get(DOMString name);
  Promise<CustomElementConstructor> whenDefined(DOMString name);
  [CEReactions] undefined upgrade(Node root);
};

dictionary ElementDefinitionOptions {
  DOMString extends;
};
```

Every `CustomElementRegistry` has a set of custom element definitions, initially empty. In general, algorithms in this specification look up elements in the registry by any of `name`, `local name`, or `constructor`.

Every `CustomElementRegistry` also has an element definition flag which is used to prevent reentrant invocations of element definition. It is initially unset.

Every `CustomElementRegistry` also has a when-defined promise map, mapping valid custom element names to promises. It is used to implement the `whenDefined()` method.

For web developers (non-normative)

```javascript
window . customElements . define(name, constructor)

Defines a new custom element, mapping the given name to the given constructor as an autonomous custom element.

window . customElements . define(name, constructor, { extends: baseLocalName })

Defines a new custom element, mapping the given name to the given constructor as a custom built-in element for the element type identified by the supplied baseLocalName. A `NotSupportedError` DOMException will be thrown upon trying to extend a custom element or an unknown element.

window . customElements . get(name)

Retrieves the custom element constructor defined for the given name. Returns undefined if there is no custom element definition with the given name.

window . customElements . whenDefined(name)

Returns a promise that will be fulfilled with the custom element's constructor when a custom element becomes defined with the given name. If such a custom element is already defined, the returned promise will be immediately fulfilled.) Returns a promise rejected with a `SyntaxError` DOMException if not given a valid custom element name.

window . customElements . upgrade(root)

Tries to upgrade all shadow-including inclusive descendant elements of root, even if they are not connected.
```

Element definition is a process of adding a custom element definition to the `CustomElementRegistry`. This is accomplished by the `define()` method. When invoked, the `define(name, constructor, options)` method must run these steps:

1. If `IsConstructor(constructor)` is false, then throw a `TypeError`.
2. If name is not a valid custom element name, then throw a `SyntaxError` DOMException.
3. If this `CustomElementRegistry` contains an entry with name name, then throw a `NotSupportedError` DOMException.
4. If this `CustomElementRegistry` contains an entry with constructor, then throw a `NotSupportedError`
DOMException.

5. Let `localName` be `name`.

6. Let `extends` be the value of the `extends` member of `options`, or null if no such member exists.

7. If `extends` is not null, then:
   1. If `extends` is a valid custom element name, then throw a "NotSupportedException" DOMException.
   2. If the `element interface` for `extends` and the `HTML namespace` is HTMLUnknownElement (e.g., if `extends` does not indicate an element definition in this specification), then throw a "NotSupportedException" DOMException.
   3. Set `localName` to `extends`.

8. If this `CustomElementRegistry`'s `element definition is running` flag is set, then throw a "NotSupportedException" DOMException.

9. Set this `CustomElementRegistry`'s `element definition is running` flag.

10. Let `formAssociated` be false.

11. Let `disableInternals` be false.

12. Let `disableShadow` be false.

13. Let `observedAttributes` be an empty sequence<DOMString>.

14. Run the following substeps while catching any exceptions:
   1. Let `prototype` be `Get(constructor, "prototype")`. Rethrow any exceptions.
   2. If `Type(prototype)` is not Object, then throw a TypeError exception.
   3. Let `lifecycleCallbacks` be a map with the keys "connectedCallback", "disconnectedCallback", "adoptedCallback", and "attributeChangedCallback", each of which belongs to an entry whose value is null.
   4. For each of the keys `callbackName` in `lifecycleCallbacks`, in the order listed in the previous step:
      1. Let `callbackValue` be `Get(prototype, callbackName)`. Rethrow any exceptions.
      2. If `callbackValue` is not undefined, then set the value of the entry in `lifecycleCallbacks` with key `callbackName` to the result of converting `callbackValue` to the Web IDL Function callback type. Rethrow any exceptions from the conversion.
   5. If the value of the entry in `lifecycleCallbacks` with key "attributeChangedCallback" is not null, then:
      1. Let `observedAttributesIterable` be `Get(constructor, "observedAttributes")`. Rethrow any exceptions.
      2. If `observedAttributesIterable` is not undefined, then set `observedAttributes` to the result of converting `observedAttributesIterable` to a sequence<DOMString>. Rethrow any exceptions from the conversion.
   6. Let `disabledFeatures` be an empty sequence<DOMString>.
   7. Let `disabledFeaturesIterable` be `Get(constructor, "disabledFeatures")`. Rethrow any exceptions.
   8. If `disabledFeaturesIterable` is not undefined, then set `disabledFeatures` to the result of converting `disabledFeaturesIterable` to a sequence<DOMString>. Rethrow any exceptions from the conversion.
   9. Set `disableInternals` to true if `disabledFeatures` contains "internals".
   10. Set `disableShadow` to true if `disabledFeatures` contains "shadow".
   11. Let `formAssociatedValue` be `Get( constructor, "formAssociated")`. Rethrow any exceptions.
   12. Set `formAssociated` to the result of converting `formAssociatedValue` to a boolean. Rethrow any exceptions from the conversion.
   13. If `formAssociated` is true, for each of "formAssociatedCallback", "formResetCallback", "formDisabledCallback", and "formStateRestoreCallback" `callbackName`:
      1. Let `callbackValue` be `Get(prototype, callbackName)`. Rethrow any exceptions.
2. If `callbackValue` is not undefined, then set the value of the entry in `lifecycleCallbacks` with key `callbackName` to the result of converting `callbackValue` to the Web IDL `Function` callback type. Rethrow any exceptions from the conversion.

Then, perform the following substep, regardless of whether the above steps threw an exception or not:

1. Unset this `CustomElementRegistry`'s element definition is running flag.

Finally, if the first set of substeps threw an exception, then rethrow that exception (thus terminating this algorithm). Otherwise, continue onward.

15. Let `definition` be a new custom element definition with `name`, `localName`, `constructor`, `observedAttributes`, `lifecycle callbacks`, `form-associated`, `disable internals`, `disableInternals`, and `disable shadow`. Then, perform the following substep, regardless of whether the above steps threw an exception or not:

1. Unset this `CustomElementRegistry`'s element definition is running flag.

Finally, if the first set of substeps threw an exception, then rethrow that exception (thus terminating this algorithm). Otherwise, continue onward.

16. Add `definition` to this `CustomElementRegistry`.

17. Let `document` be this `CustomElementRegistry`'s relevant global object's associated Document.

18. Let `upgrade candidates` be all elements that are shadow-including descendants of `document`, whose namespace is the HTML namespace and whose local name is `localName`, in shadow-including tree order. Additionally, if `extends` is non-null, only include elements whose `is value` is equal to `name`.

19. For each element `element` in `upgrade candidates`, `enqueue a custom element upgrade reaction` given `element` and `definition`.

20. If this `CustomElementRegistry`'s when-defined promise map contains an entry with key `name`:

   1. Let `promise` be the value of that entry.
   2. Resolve `promise` with `constructor`.
   3. Delete the entry with key `name` from this `CustomElementRegistry`'s when-defined promise map.

When invoked, the `get(name)` method must run these steps:

1. If this `CustomElementRegistry` contains an entry with `name`, then return that entry's `constructor`.

2. Otherwise, return undefined.

When invoked, the `whenDefined(name)` method must run these steps:

1. If `name` is not a valid custom element name, then return a new promise rejected with a "SyntaxError" DOMException.

2. If this `CustomElementRegistry` contains an entry with `name`, then return a new promise resolved with that entry's `constructor`.

3. Let `map` be this `CustomElementRegistry`'s when-defined promise map.

4. If `map` does not contain an entry with key `name`, create an entry in `map` with key `name` and whose value is a new promise.

5. Let `promise` be the value of the entry in `map` with key `name`.

6. Return `promise`.

Example

The `whenDefined()` method can be used to avoid performing an action until all appropriate custom elements are defined. In this example, we combine it with the :defined pseudo-class to hide a dynamically-loaded article's contents until we're sure that all of the autonomous custom elements it uses are defined.

```javascript
articleContainer.hidden = true;
fetch(articleURL)
  .then(response => response.text())
  .then(text => {
    articleContainer.innerHTML = text;
  });
```
When invoked, the `upgrade(root)` method must run these steps:

1. Let `candidates` be a list of all of root's shadow-including inclusive descendant elements, in shadow-including tree order.
2. For each candidate of candidates, try to upgrade candidate.

Example

The `upgrade()` method allows upgrading of elements at will. Normally elements are automatically upgraded when they become connected, but this method can be used if you need to upgrade before you’re ready to connect the element.

```javascript
const el = document.createElement("spider-man");

class SpiderMan extends HTMLElement {

    constructor() {
        super();
        const b = document.querySelector("#b");
        b.remove();
        console.assert(!el instanceof SpiderMan); // not yet upgraded
    }

customElements.upgrade(el);

customElements.define("spider-man", SpiderMan);

console.assert(el instanceof SpiderMan); // upgraded!
```

4.13.5 Upgrades

To upgrade an element, given as input a `custom element definition` definition and an element `element`, run the following steps:

1. If element's `custom element state` is not "undefined" or "uncustomized", then return.

Example

One scenario where this can occur due to reentrant invocation of this algorithm, as in the following example:

```html
<!DOCTYPE html>
<x-foo id="a"></x-foo>
<x-foo id="b"></x-foo>

<script>
// Defining enqueues upgrade reactions for both "a" and "b"
customElements.define("x-foo", class extends HTMLElement {
    constructor() {
        super();

        const b = document.querySelector("#b");
        b.remove();

        // While this constructor is running for "a", "b" is still undefined, and so inserting it into the document will enqueue a second upgrade reaction for "b" in addition to the one enqueued by defining x-foo.
        document.body.appendChild(b);
```
This step will thus bail out the algorithm early when `upgrade an element` is invoked with "b" a second time.

2. Set element's `custom element definition` to `definition`.

3. Set element's `custom element state` to "failed".

**Note**

It will be set to "custom" after the upgrade succeeds. For now, we set it to "failed" so that any reentrant invocations will hit the above early-exit step.

4. For each `attribute` in element's `attribute list`, in order, enqueue a `custom element callback reaction` with element, callback name "attributeChangedCallback", and an argument list containing attribute's local name, null, attribute's value, and attribute's namespace.

5. If `element` is `connected`, then enqueue a `custom element callback reaction` with element, callback name "connectedCallback", and an empty argument list.

6. Add `element` to the end of `definition's construction stack`.

7. Let `C` be `definition's constructor`.

8. Run the following substeps while catching any exceptions:

   1. If `definition's disable shadow` is true and `element's shadow root` is non-null, then throw a "NotSupportedError" `DOMException`.

   **Note**

   This is needed as `attachShadow()` does not use look up a `custom element definition` while `attachInternals()` does.

   2. Set element's `custom element state` to "precustomized".

   3. Let `constructResult` be the result of constructing `C`, with no arguments.

   **Note**

   If `C` non-conformantly uses an API decorated with the [CEReactions] extended attribute, then the reactions enqueued at the beginning of this algorithm will execute during this step, before `C` finishes and control returns to this algorithm. Otherwise, they will execute after `C` and the rest of the upgrade process finishes.

   4. If `SameValue(constructResult, element)` is false, then throw a `TypeError`.

   **Note**

   This can occur if `C` constructs another instance of the same custom element before calling `super()`, or if `C` uses JavaScript's `return-override` feature to return an arbitrary `HTMLElement` object from the constructor.

Then, perform the following substep, regardless of whether the above steps threw an exception or not:

1. Remove the last entry from the end of `definition's construction stack`.

**Note**

Assuming `C` calls `super()` (as it will if it is conformant), and that the call succeeds, this will be the already constructed marker that replaced the element we pushed at the beginning of this algorithm. (The `HTML element constructor` carries out this replacement.)

If `C` does not call `super()` (i.e. it is not conformant), or if any step in the `HTML element constructor` throws, then this entry will still be element.

Finally, if the above steps threw an exception, then:
1. Set element's custom element definition to null.
2. Empty element's custom element reaction queue.
3. Rethrow the exception (thus terminating this algorithm).

9. If element is a form-associated custom element, then:
   1. Reset the form owner of element. If element is associated with a form element, then enqueue a custom element callback reaction with element, callback name "formAssociatedCallback", and « the associated form ».
   2. If element is disabled, then enqueue a custom element callback reaction with element, callback name "formDisabledCallback" and « true ».
10. Set element's custom element state to "custom".

To try to upgrade an element, given as input an element element, run the following steps:

1. Let definition be the result of looking up a custom element definition given element's node document, element's namespace, element's local name, and element's is value.
2. If definition is not null, then enqueue a custom element upgrade reaction given element and definition.

4.13.6 Custom element reactions

A custom element possesses the ability to respond to certain occurrences by running author code:

- When upgraded, its constructor is run, with no arguments.
- When it becomes connected, its connectedCallback is called, with no arguments.
- When it becomes disconnected, its disconnectedCallback is called, with no arguments.
- When it is adopted into a new document, its adoptedCallback is called, given the old document and new document as arguments.
- When any of its attributes are changed, appended, removed, or replaced, its attributeChangedCallback is called, given the attribute's local name, old value, new value, and namespace as arguments. (An attribute's old or new value is considered to be null when the attribute is added or removed, respectively.)
- When the user agent resets the form owner of a form-associated custom element, and doing so changes the form owner, its formAssociatedCallback is called, given the new form owner (or null if no owner) as an argument.
- When the form owner of a form-associated custom element is reset, its formResetCallback is called.
- When the disabled state of a form-associated custom element is changed, its formDisabledCallback is called, given the new state as an argument.
- When user agent updates a form-associated custom element's value on behalf of a user, its formStateRestoreCallback is called, given the new value and a string indicating a reason, "restore" or "autocomplete", as arguments.

We call these reactions collectively custom element reactions.

The way in which custom element reactions are invoked is done with special care, to avoid running author code during the middle of delicate operations. Effectively, they are delayed until "just before returning to user script". This means that for most purposes they appear to execute synchronously, but in the case of complicated composite operations (like cloning, or range manipulation), they will instead be delayed until after all the relevant user agent processing steps have completed, and then run together as a batch.

Additionally, the precise ordering of these reactions is managed via a somewhat-complicated stack-of-queues system, described below. The intention behind this system is to guarantee that custom element reactions always are invoked in the same order as their triggering actions, at least within the local context of a single custom element. (Because custom element reaction code can
Each similar-origin window agent\(^{p899}\) has a custom element reactions stack, which is initially empty. A similar-origin window agent\(^{p899}\)'s current element queue is the element queue\(^{p713}\) at the top of its custom element reactions stack\(^{p713}\). Each item in the stack is an element queue, which is initially empty as well. Each item in an element queue\(^{p713}\) is an element. (The elements are not necessarily custom yet, since this queue is used for upgrades\(^{p710}\) as well.)

Each custom element reactions stack\(^{p713}\) has an associated backup element queue, which an initially-empty element queue\(^{p713}\). Elements are pushed onto the backup element queue\(^{p713}\) during operations that affect the DOM without going through an API decorated with [CEReactions]\(^{p714}\), or through the parser's create an element for the token\(^{p1119}\) algorithm. An example of this is a user-initiated editing operation which modifies the descendants or attributes of an editable element. To prevent reentrancy when processing the backup element queue\(^{p713}\), each custom element reactions stack\(^{p713}\) also has a processing the backup element queue flag, initially unset.

All elements have an associated custom element reaction queue, initially empty. Each item in the custom element reaction queue\(^{p713}\) is of one of two types:

- An upgrade reaction, which will upgrade\(^{p710}\) the custom element and contains a custom element definition\(^{p706}\); or
- A callback reaction, which will call a lifecycle callback, and contains a callback function as well as a list of arguments.

This is all summarized in the following schematic diagram:

To enqueue an element on the appropriate element queue, given an element element, run the following steps:

1. Let reactionsStack be element's relevant agent\(^{p900}\)'s custom element reactions stack\(^{p713}\).
2. If reactionsStack is empty, then:
   1. Add element to reactionsStack's backup element queue\(^{p713}\).
   2. If reactionsStack's processing the backup element queue\(^{p713}\) flag is set, then return.
   3. Set reactionsStack's processing the backup element queue\(^{p713}\) flag.
   4. Queue a microtask\(^{p933}\) to perform the following steps:
      1. Invoke custom element reactions\(^{p714}\) in reactionsStack's backup element queue\(^{p713}\).
2. Unset $\text{reactionsStack}$'s processing the backup element queue\(^{713}\) flag.

3. Otherwise, add element to element's relevant agent\(^{900}\)'s current element queue\(^{713}\).

To enqueue a custom element callback reaction, given a custom element\(^{704}\) element, a callback name $\text{callbackName}$, and a list of arguments $\text{args}$, run the following steps:

1. Let $\text{definition}$ be element's custom element definition.
2. Let $\text{callback}$ be the value of the entry in $\text{definition}$'s lifecycle callbacks\(^{706}\) with key $\text{callbackName}$.
3. If $\text{callback}$ is null, then return
4. If $\text{callbackName}$ is "attributeChangedCallback", then:
   1. Let $\text{attributeName}$ be the first element of $\text{args}$.
   2. If $\text{definition}$'s observed attributes\(^{706}\) does not contain $\text{attributeName}$, then return.
5. Add a new $\text{callback reaction}$\(^{713}\) to element's custom element reaction queue\(^{713}\), with callback function $\text{callback}$ and arguments $\text{args}$.
6. Enqueue an element on the appropriate element queue\(^{713}\) given element.

To enqueue a custom element upgrade reaction, given an element $\text{element}$ and custom element definition\(^{706}\) $\text{definition}$, run the following steps:

1. Add a new $\text{upgrade reaction}$\(^{713}\) to element's custom element reaction queue\(^{713}\), with custom element definition\(^{706}\) $\text{definition}$.
2. Enqueue an element on the appropriate element queue\(^{713}\) given element.

To invoke custom element reactions in an element queue\(^{713}\) queue, run the following steps:

1. For each custom element\(^{704}\) element in queue:
   1. Let $\text{reactions}$ be element's custom element reaction queue\(^{713}\).
   2. Repeat until $\text{reactions}$ is empty:
      1. Remove the first element of $\text{reactions}$, and let $\text{reaction}$ be that element. Switch on $\text{reaction}$'s type:
         - $\text{upgrade reaction}$\(^{713}\)
         Upgrade\(^{710}\) element using reaction's custom element definition\(^{706}\).
         - $\text{callback reaction}$\(^{713}\)
         Invoke reaction's callback function with reaction's arguments, and with $\text{element}$ as the callback this value.
      If this throws an exception, catch it, and report the exception\(^{923}\).

To ensure custom element reactions\(^{712}\) are triggered appropriately, we introduce the [CEReactions] IDL extended attribute. It indicates that the relevant algorithm is to be supplemented with additional steps in order to appropriately track and invoke custom element reactions\(^{712}\).

The [CEReactions]\(^{714}\) extended attribute must take no arguments, and must not appear on anything other than an operation, attribute, setter, or deleter. Additionally, it must not appear on readonly attributes.

Operations, attributes, setters, or deleters annotated with the [CEReactions]\(^{714}\) extended attribute must run the following steps in place of the ones specified in their description:

1. Push a new element queue\(^{713}\) onto this object's relevant agent\(^{900}\)'s custom element reactions stack\(^{713}\).
2. Run the originally-specified steps for this construct, catching any exceptions. If the steps return a value, let $\text{value}$ be the returned value. If they throw an exception, let $\text{exception}$ be the thrown exception.
3. Let $\text{queue}$ be the result of popping from this object's relevant agent\(^{900}\)'s custom element reactions stack\(^{713}\).
4. **Invoke custom element reactions** in queue.

5. If an exception was thrown by the original steps, rethrow exception.

6. If a value was returned from the original steps, return value.

---

**Note**

The intent behind this extended attribute is somewhat subtle. One way of accomplishing its goals would be to say that every operation, attribute, setter, and deleter on the platform must have these steps inserted, and to allow implementers to optimize away unnecessary cases (where no DOM mutation is possible that could cause custom element reactions to occur).

However, in practice this imprecision could lead to non-interoperable implementations of custom element reactions, as some implementations might forget to invoke these steps in some cases. Instead, we settled on the approach of explicitly annotating all relevant IDL constructs, as a way of ensuring interoperable behavior and helping implementations easily pinpoint all cases where these steps are necessary.

Any nonstandard APIs introduced by the user agent that could modify the DOM in such a way as to cause enqueuing a custom element callback reaction or enqueuing a custom element upgrade reaction, for example by modifying any attributes or child elements, must also be decorated with the [CEReactions] attribute.

---

**Note**

As of the time of this writing, the following nonstandard or not-yet-standardized APIs are known to fall into this category:

- `HTMLElement`'s `outerText` IDL attribute
- `HTMLInputElement`'s `webkitdirectory` and `incremental` IDL attributes
- `HTMLLinkElement`'s `scope` IDL attribute
- `ShadowRoot`'s `innerHTML` IDL attribute

---

### 4.13.7 Element internals

Certain capabilities are meant to be available to a custom element author, but not to a custom element consumer. These are provided by the `element.attachInternals()` method, which returns an instance of `ElementInternals`. The properties and methods of `ElementInternals` allow control over internal features which the user agent provides to all elements.

For web developers (non-normative)

`element.attachInternals()`

Returns an `ElementInternals` object targeting the custom element element. Throws an exception if `element` is not a custom element, if the "internals" feature was disabled as part of the element definition, or if it is called twice on the same element.

Each `HTMLElement` has an attached internals boolean, initially false.

The `attachInternals()` method steps are:

1. If `this.is value` is not null, then throw a "NotSupportedError" DOMException.
2. Let `definition` be the result of looking up a custom element definition given this's node document, its namespace, its local name, and null as the `is value`.
3. If `definition` is null, then throw an "NotSupportedError" DOMException.
4. If `definition's disable internals` is true, then throw a "NotSupportedError" DOMException.
5. If `this's attached internals` is true, then throw an "NotSupportedError" DOMException.
6. If `this's custom element state` is not "precustomized" or "custom", then throw a "NotSupportedError" DOMException.
7. Set `this's attached internals` to true.
8. Return a new `ElementInternals` instance whose `target element` is `this`.

### 4.13.7.1 The `ElementInternals` interface

The IDL for the `ElementInternals` interface is as follows, with the various operations and attributes defined in the following sections:

```idl
[Exposed=Window]
interface ElementInternals {
  // Shadow root access
  readonly attribute ShadowRoot? shadowRoot;

  // Form-associated custom elements
  undefined setValue((File or USVString or FormData)? value,
                      optional (File or USVString or FormData)? state);
  readonly attribute HTMLFormElement? form;

  undefined setValidity(optional ValidityStateFlags flags = {},
                        optional DOMString message,
                        optional HTMLElement anchor);
  readonly attribute boolean willValidate;
  readonly attribute ValidityState validity;
  readonly attribute DOMString validationMessage;
  boolean checkValidity();
  boolean reportValidity();

  readonly attribute NodeList labels;
};
```

Each `ElementInternals` has a `target element`, which is a `custom element`.

### 4.13.7.2 Shadow root access

For web developers (non-normative)

```plaintext
internals . shadowRoot
```

Returns the `ShadowRoot` for `internals's target element`, if the `target element` is a `shadow host`, or null otherwise.

The `shadowRoot` getter steps are:

1. Let target be `this's target element`.
2. If target is not a shadow host, then return null.
3. Let shadow be target's shadow root.
4. If shadow's available to element internals is false, then return null.
5. Return shadow.

4.13.7.3 Form-associated custom elements

For web developers (non-normative)

`internals . setFormValue(value)`

Sets both the state and submission value of internals's target element to value.
If value is null, the element won't participate in form submission.

`internals . setFormValue(value, state)`

Sets the submission value of internals's target element to value, and its state to state.
If value is null, the element won't participate in form submission.

`internals . form` Returns the form owner of internals's target element.

`internals . setValidity(flags, message [, anchor])`

Marks internals's target element as suffering from the constraints indicated by the flags argument, and sets the element's validation message to message. If anchor is specified, the user agent might use it to indicate problems with the constraints of internals's target element when the form owner is validated interactively or reportValidity() is called.

`internals . setValidity({})` Marks internals's target element as satisfying its constraints.

`internals . willValidate` Returns true if internals's target element will be validated when the form is submitted; false otherwise.

`internals . validity` Returns the ValidityState object for internals's target element.

`internals . validationMessage` Returns the error message that would be shown to the user if internals's target element was to be checked for validity.

`valid = internals . checkValidity()` Returns true if internals's target element has no validity problems; false otherwise. Fires an invalid event at the element in the latter case.

`valid = internals . reportValidity()` Returns true if internals's target element 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.

`internals . labels` Returns a NodeList of all the label elements that internals's target element is associated with.

Each form-associated custom element has submission value. It is used to provide one or more entries on form submission. The initial value of submission value is null, and submission value can be null, a string, a File, or a list of entries.

Each form-associated custom element has state. It is information with which the user agent can restore a user's input for the element. The initial value of state is null, and state can be null, a string, a File, or a list of entries.

The setFormValue() method is used by the custom element author to set the element's submission value and state, thus communicating these to the user agent.

When the user agent believes it is a good idea to restore a form-associated custom element's state, for example after navigation or restarting the user agent, they may enqueue a custom element callback reaction with that element, callback name "formStateRestoreCallback", and an argument list containing the state to be restored, and "restore".
If the user agent has a form-filling assist feature, then when the feature is invoked, it may enqueue a custom element callback reaction with a form-associated custom element callback name "formStateRestoreCallback", and an argument list containing the state value determined by history of state value and some heuristics, and "autocomplete".

In general, the state is information specified by a user, and the submission value is a value after canonicalization or sanitization, suitable for submission to the server. The following examples makes this concrete:

Example
Suppose that we have a form-associated custom element which asks a user to specify a date. The user specifies "3/15/2019", but the control wishes to submit "2019-03-15" to the server. "3/15/2019" would be a state of the element, and "2019-03-15" would be a submission value.

Example
Suppose you develop a custom element emulating a the behavior of the existing checkbox,input type. Its submission value would be the value of its value content attribute, or the string "on". Its state would be one of "checked", "unchecked", "checked/indeterminate", or "unchecked/indeterminate".

The setFormValue(value, state) method steps are:

1. Let element be this's target element.
2. If element is not a form-associated custom element, then throw a NotSupportedError DOMException.
3. Set target element's submission value to value if value is not a FormData object, or to a clone of the entry list associated with value otherwise.
4. If the state argument of the function is omitted, set element's state to its submission value.
5. Otherwise, if state is a FormData object, set element's state to clone of the entry list associated with state.
6. Otherwise, set element's state to state.

Each form-associated custom element has validity flags named valueMissing, typeMismatch, patternMismatch, tooLong, tooShort, rangeUnderflow, rangeOverflow, stepMismatch, and customError. They are false initially.

Each form-associated custom element has a validation message string. It is the empty string initially.

Each form-associated custom element has a validation anchor element. It is null initially.

The setValidity(flags, message, anchor) method steps are:

1. Let element be this's target element.
2. If element is not a form-associated custom element, then throw a NotSupportedError DOMException.
3. If flags contains one or more true values and message is not given or is the empty string, then throw a TypeError.
4. For each entry flag → value of flags, set element's validity flag with the name flag to value.
5. Set element's validation message to the empty string if message is not given or all of element's validity flags are false, or to message otherwise.
6. If element's customError validity flag is true, then set element's custom validity error message to element's validation message. Otherwise, set element's custom validity error message to the empty string.
7. Set element's validation anchor to null if anchor is not given. Otherwise, if anchor is not a shadow-including descendant of element, then throw a NotFoundError DOMException. Otherwise, set element's validation anchor to anchor.

The validationMessage getter steps are to return the validation message of this's target element.

The entry construction algorithm for a form-associated custom element, given an element element and a list entry list, consists of the following steps:

1. If element's submission value is a list of entries, then append each item of element's submission value to entry list,
and return.

Note

In this case, user agent does not refer to the name\(^\text{p568}\) content attribute value. An implementation of form-associated custom element\(^\text{p705}\) is responsible to decide names of entries. They can be the name\(^\text{p568}\) content attribute value, they can be strings based on the name\(^\text{p568}\) content attribute value, or they can be unrelated to the name\(^\text{p568}\) content attribute.

2. If the element does not have a name\(^\text{p568}\) attribute specified, or its name\(^\text{p568}\) attribute's value is the empty string, then return.

3. If the element's submission value\(^\text{p717}\) is not null, append an entry\(^\text{p601}\) to entry list with the name\(^\text{p568}\) attribute value and the submission value\(^\text{p717}\).

4.13.7.4 Accessibility semantics \(^\text{p71}\)

For web developers (non-normative)

```
# Role

\(\text{internals} \cdot \text{role}\[^\text{p63}\] \ [\text{= value}]\)

Sets or retrieves the default ARIA role for \(\text{internals}^\text{'s} \) target element\(^\text{p716}\), which will be used unless the page author overrides it using the role\(^\text{p63}\) attribute.

\(\text{internals} \cdot \text{aria-*}\[^\text{p63}\] \ [\text{= value}]\)

Sets or retrieves various default ARIA states or property values for \(\text{internals}^\text{'s} \) target element\(^\text{p716}\), which will be used unless the page author overrides them using the aria-*\[^\text{p63}\] attributes.
```

Each custom element\(^\text{p708}\) has a native accessibility semantics map, which is a map\(^\text{p442}\), initially empty. See the Requirements related to ARIA and to platform accessibility APIs\(^\text{p152}\) section for information on how this impacts platform accessibility APIs.

ElementInternals\(^\text{p716}\) includes the ARIAMixin mixin. The accessors provided by this mixin are used to manipulate the target element\(^\text{p716}\)’s native accessibility semantics map\(^\text{p719}\), as follows:

The ARIAMixin getter steps for ElementInternals\(^\text{p716}\), given internals, idlAttribute, and contentAttribute, are:

1. Let map be internals’s target element\(^\text{p716}\)’s native accessibility semantics map\(^\text{p719}\).
2. If map[contentAttribute] exists, then return it.
3. Return null.

The ARIAMixin setter steps for ElementInternals\(^\text{p716}\), given internals, idlAttribute, contentAttribute, and value, are:

1. Let map be internals's target element\(^\text{p716}\)’s native accessibility semantics map\(^\text{p719}\).
2. If value is null, then remove map[contentAttribute].
3. Otherwise, set map[contentAttribute] to value.

4.14 Common idioms without dedicated elements \(^\text{p71}\)

4.14.1 Bread crumb navigation \(^\text{p71}\)

This specification does not provide a machine-readable way of describing bread-crumb navigation menus. Authors are encouraged to just use a series of links in a paragraph. The nav\(^\text{p186}\) element can be used to mark the section containing these paragraphs as being navigation blocks.

Example

In the following example, the current page can be reached via two paths.

```
<nav>
  <p>
```

719
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**

Here, three tags are included in a short tag cloud:

```html
<style>
.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; }
@media speech {
 .tag-cloud > li > span { display:inline }
}
</style>

<ul class="tag-cloud">
<li class="tag-cloud-4"><a title="28 instances" href="/t/apple">apple</a> <span>(popular)</span></li>
<li class="tag-cloud-2"><a title="6 instances" href="/t/kiwi">kiwi</a> <span>(rare)</span></li>
<li class="tag-cloud-5"><a title="41 instances" href="/t/pear">pear</a> <span>(very popular)</span></li>
</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.

---

4.14.3 Conversations

This specification does not define a specific element for marking up conversations, meeting minutes, chat transcripts, dialogues 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.

This example demonstrates this using an extract from Abbot and Costello's famous sketch, *Who's on first*:

```html
<p>Costello: Look, you gotta first baseman?</p>
<p>Abbott: Certainly.</p>
<p>Costello: Who's playing first?</p>
<p>Abbott: That's right.</p>
<p>Costello becomes exasperated.</p>
<p>Costello: When you pay off the first baseman every month, who gets the money?</p>
<p>Abbott: Every dollar of it.
```

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.

```html
<p data-value="1319898155">14:22</p> <b>egof</b> I'm not that nerdy, I've only seen 30% of the star trek episodes
<p data-value="1319898192">14:23</p> <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</p> <b>egof</b> it's unarguably
<p data-value="1319898228">14:23</p> <i>* kaj blinks</i>
<p data-value="1319898260">14:24</p> <b>kaj</b> you are not helping your case
```

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]

```html
<p>Next, you meet a fisher. You can say one of several greetings:
<dl>
<dt>"Hello there!"
<dd><p>She responds with "Hello, how may I help you?"; you can respond with:
<dl>
<dt>"I would like to buy a fish."
<dd><p>She sells you a fish and the conversation finishes.
<dt>"Can I borrow your boat?"
<dd><p>She 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>She lends you her 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", she replies. Your conversation options
```

721
at this point are the same as they were after asking to borrow her boat, minus any options you've suggested before.

"Vote for me in the next election!"
She turns away. The conversation finishes.
"Madam, are you aware that your fish are running away?"
She looks at you skeptically and says "Fish cannot run, miss".
"You got me!"
The fisher sighs and the conversation ends.
"Only kidding."
"Good one!" she retorts. Your conversation options at this point are the same as those following "Hello there!" above.
"Oh, then what are they doing?"
She looks at her fish, giving you an opportunity to steal her boat, which you do. The conversation ends.

Example

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:

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. The Shopkeeper

The Shopkeeper

I'm about to fly out, sorry!
Sorry, I'm just waiting for flight clearance and then I'll be off!

Before the accident:

After the accident:

I'm about to fly out, sorry!
Ok, I'm not leaving right now, my plane is being cleaned.
Ok, it's not being cleaned, it needs a minor repair first.
Ok, ok, stop bothering me! Truth is, I had a crash.

The pilot

During the first clan meeting:

Hey, have you seen my daughter? I bet she's up to something nefarious again...
Nice weather we're having today, eh?
The name is Bailey, Jeff Bailey. How can I help you today?
A glass of water? Fresh from the well!
4.14.4 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**

In this example, two parts of a dialogue are annotated with footnote-like content using the `title` attribute.

```html
<p><b>Customer</b>: Hello! I wish to register a complaint. Hello. Miss?</p>
<p><b>Shopkeeper</b>: <span title="Colloquial pronunciation of 'What do you'">
Watcha</span> mean, miss?</p>
<p><b>Customer</b>: Uh, I'm sorry, I have a cold. I wish to make a complaint.</p>
<p><b>Shopkeeper</b>: Sorry, <span title="This is, of course, a lie.">we're closing for lunch</span>.</p>
```

**Note**

Unfortunately, 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).

**Note**

If the `title` attribute is used, CSS can be used to draw the reader's attention to the elements with the attribute.

**Example**

For example, the following CSS places a dashed line below elements that have a `title` attribute.

```css
<title> { border-bottom: thin dashed; }
```

For longer 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**

In this example, a footnote in the dialogue links to a paragraph below the dialogue. The paragraph then reciprocally links back to the dialogue, allowing the user to return to the location of the footnote.

```html
<p>Announcer: Number 16: The <i>hand</i>.</p>
<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>
<p>Norman: I don't. <sup><a href="#fn1" id="r1">[1]</a></sup></p>
```
Interviewer: You told me you did!
...

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.

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

In this example, a sidebar is given after a dialogue, giving it some context.

Example

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.
An element is said to be **actually disabled** if it is one of the following:

- 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 `fieldset` element that is a disabled fieldset
- a form-associated custom element that is disabled

**Note**

This definition is used to determine what elements are focusable and which elements match the `.enabled` and `.disabled` pseudo classes.

### 4.16 Matching HTML elements using selectors and CSS

#### 4.16.1 Case-sensitivity of the CSS `attr()` function

CSS Values and Units leaves the case-sensitivity of attribute names for the purpose of the `attr()` function to be defined by the host language. [CSSVALUES][1272]

When comparing the attribute name part of a CSS `attr()` function to the names of namespace-less attributes on HTML elements in HTML documents, the name part of the CSS `attr()` function must first be converted to ASCII lowercase. The same function when compared to other attributes must be compared according to its original case. In both cases, to match the values must be identical to each other (and therefore the comparison is case sensitive).

**Note**

This is the same as comparing the name part of a CSS attribute selector, specified in the next section.

#### 4.16.2 Case-sensitivity of selectors

Selectors leaves the case-sensitivity of element names, attribute names, and attribute values to be defined by the host language. [SELECTORS][1276]

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, to match the values must be identical to each other (and therefore the comparison is case sensitive).

When comparing the name part of a CSS attribute selector to the names of 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:

- accept
- accept-charset
- align
- alink
- axis
- bgcolor
There are a number of dynamic selectors that can be used with HTML. This section defines when these selectors match HTML elements. [SELECTORS][1276] [CSSUI][31272]

`:defined`

The `:defined` pseudo-class must match any element that is `defined`.

Note

Selectors defines that ID and class selectors (such as `#foo` and `.bar`), when matched against elements in documents that are in quirks mode, will be matched in an ASCII case-insensitive manner. However, this does not apply for attribute selectors with "id" or "class" as the name part. The selector `[class="foobar"]` will treat its value as case-sensitive even in quirks mode.
All `a` elements that have an `href` attribute, and all `area` 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.

  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, which ever comes first.

- If the element is focusable
  The element is being actively pointed at 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 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 finger in a multitouch environment, while the finger is touching the display surface).

`: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**

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 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 is focusable
  The element is being actively pointed at 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 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 finger in a multitouch environment, while the finger is touching the display surface).
For the purposes of the CSS \(\text{:focus}\) pseudo-class, an element has the focus when:

- it is not itself a browsing context container\(^{914}\); and
- at least one of the following is true:
  - it is one of the elements listed in the current focus chain of the top-level browsing context\(^{973}\), or
  - its shadow root, shadowRoot, is not null and shadowRoot is the root of at least one element that has the focus\(^{928}\).

For the purposes of the CSS \(\text{:target}\) pseudo-class, the Document's target elements are a list containing the Document's target element\(^{909}\), if it is not null, or containing no elements, if it is. [SELECTORS]\(^{915}\).

The \(\text{:enabled}\) pseudo-class must match any button, input, select, textarea, optgroup, option, fieldset element, or form-associated custom element that is not actually disabled\(^{925}\).

The \(\text{:disabled}\) pseudo-class must match any element that is actually disabled\(^{925}\).

The \(\text{:checked}\) 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

The \(\text{:indeterminate}\) 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 elements whose checkedness state is true.
- progress elements with no value content attribute

The \(\text{:default}\) pseudo-class must match any element falling into one of the following categories:

- Submit buttons that are default buttons of their form owner.
- input elements to which the checked attribute applies and that have a checked attribute
- option elements that have a selected attribute

The \(\text{:placeholder-shown}\) pseudo-class must match any element falling into one of the following categories:
• `input` elements that have a `placeholder` attribute whose value is currently being presented to the user.

• `textarea` elements that have a `placeholder` attribute whose value is currently being presented to the user.

::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 validation 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

::autofill

::-webkit-autofill

The ::autofill and ::-webkit-autofill pseudo-classes must match `input` elements which have been autofilled by user agent. These pseudo-classes must stop matching if the user edits the autofilled field.

Note

One way such autofilling might happen is via the autocomplete attribute, but user agents could autofill even without that attribute being involved.

::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: [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**

*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 Selectors.* [SELECTORS]
5 Microdata

5.1 Introduction

5.1.1 Overview

This section is non-normative.

Sometimes, it is desirable to annotate content with specific machine-readable labels, e.g. to allow generic scripts to provide services that are customized to the page, or to enable content from a variety of cooperating authors to be processed by a single script in a consistent manner.

For this purpose, authors can use the microdata features described in this section. Microdata allows nested groups of name-value pairs to be added to documents, in parallel with the existing content.

5.1.2 The basic syntax

This section is non-normative.

At a high level, microdata consists of a group of name-value pairs. The groups are called items, and each name-value pair is a property. Items and properties are represented by regular elements.

To create an item, the itemscope attribute is used.

To add a property to an item, the itemprop attribute is used on one of the item's descendants.

Example

Here there are two items, each of which has the property "name":

```html
<div itemscope>
  <p>My name is <span itemprop="name">Elizabeth</span>.</p>
</div>

<div itemscope>
  <p>My name is <span itemprop="name">Daniel</span>.</p>
</div>
```

Markup without the microdata-related attributes does not have any effect on the microdata model.

Example

These two examples are exactly equivalent, at a microdata level, as the previous two examples respectively:

```html
<div itemscope>
  <p>My name is <span itemprop="name">Elizabeth</span>.</p>
</div>

<section>
  <div itemscope>
    <aside>
      <p>My name is <span itemprop="name"><a href="/?user=daniel">Daniel</a></span>.</p>
    </aside>
  </div>
</section>
```

Properties generally have values that are strings.
Here the item has three properties:

```html
<div itemscope>
<p>My name is <span itemprop="name">Neil</span>.</p>
<p>My band is called <span itemprop="band">Four Parts Water</span>.</p>
<p>I am <span itemprop="nationality">British</span>.</p>
</div>
```

When a string value is a URL, it is expressed using the `<a>` element and its `href` attribute, the `<img>` element and its `src` attribute, or other elements that link to or embed external resources.

For numeric data, the `<meter>` element and its `value` attribute can be used instead.

Similarly, for date- and time-related data, the `<time>` element and its `datetime` attribute can be used instead.

Properties can also themselves be groups of name-value pairs, by putting the `itemscope` attribute on the element that declares the property.
Items that are not part of others are called top-level microdata items.\textsuperscript{734}

Example

In this example, the outer item represents a person, and the inner one represents a band:

\[
\text{<div itemscope>}
\text{<p> Name: <span itemprop="name">Amanda</span></p>}
\text{<p> Band: <span itemprop="band" itemscope><span itemprop="name">Jazz Band</span> (12 players)</span></p>}
\text{</div>}
\]

The outer item here has two properties, "name" and "band". The "name" is "Amanda", and the "band" is an item in its own right, with two properties, "name" and "size". The "name" of the band is "Jazz Band", and the "size" is "12".

Properties that are not descendants of the element with the \textit{itemscope} attribute can be associated with the \textit{item} using the \textit{itemref} attribute. This attribute takes a list of IDs of elements to crawl in addition to crawling the children of the element with the \textit{itemscope} attribute.

Example

This example is the same as the previous one, but all the properties are separated from their items.

\[
\text{<div itemscope id="amanda" itemref="a b">}
\text{<p id="a"> Name: <span itemprop="name">Amanda</span></p>}
\text{<div id="b" itemprop="band" itemscope itemref="c">}
\text{<div id="c">}
\text{<p> Band: <span itemprop="name">Jazz Band</span></p>}
\text{<p> Size: <span itemprop="size">12</span> players</p>}
\text{</div>}
\text{</div>}
\]

This gives the same result as the previous example. The first item has two properties, "name", set to "Amanda", and "band", set to another item. That second item has two further properties, "name", set to "Jazz Band", and "size", set to "12".

An \textit{item} can have multiple properties with the same name and different values.

Example

This example describes an ice cream, with two flavors:

\[
\text{<div itemscope>}
\text{<p> Flavors in my favorite ice cream:</p>}
\text{<ul>}
\text{<li itemprop="flavor">Lemon sorbet</li>}
\text{<li itemprop="flavor">Apricot sorbet</li>}
\text{</ul>}
\text{</div>}
\]

This thus results in an item with two properties, both "flavor", having the values "Lemon sorbet" and "Apricot sorbet".

An element introducing a property can also introduce multiple properties at once, to avoid duplication when some of the properties have the same value.

Example

Here we see an item with two properties, "favorite-color" and "favorite-fruit", both set to the value "orange":

\[
\text{<div itemscope>}
\text{<span itemprop="favorite-color favorite-fruit">orange</span>}
\text{</div>}
\]
It's important to note that there is no relationship between the microdata and the content of the document where the microdata is marked up.

**Example**

There is no semantic difference, for instance, between the following two examples:

```html
<figure>
  <img src="castle.jpeg">
  <figcaption><span itemscope><span itemprop="name">The Castle</span> (1986)</span></figcaption>
</figure>

<figure>
  <img src="castle.jpeg">
  <figcaption>The Castle (1986)</figcaption>
</figure>
```

Both have a figure with a caption, and both, completely unrelated to the figure, have an item with a name-value pair with the name "name" and the value "The Castle". The only difference is that if the user drags the caption out of the document, in the former case, the item will be included in the drag-and-drop data. In neither case is the image in any way associated with the item.

### 5.1.3 Typed items

This section is non-normative.

The examples in the previous section show how information could be marked up on a page that doesn't expect its microdata to be reused. Microdata is most useful, though, when it is used in contexts where other authors and readers are able to cooperate to make new uses of the markup.

For this purpose, it is necessary to give each item a type, such as "https://example.com/person", or "https://example.org/cat", or "https://band.example.net/". Types are identified as URLs.

The type for an item is given as the value of an `itemtype` attribute on the same element as the `itemscope` attribute.

**Example**

Here, the item's type is "https://example.org/animals#cat"

```html
<section itemscope itemtype="https://example.org/animals#cat">
  <h1 itemprop="name">Hedral</h1>
  <p itemprop="desc">Hedral is a male american domestic shorthair, with a fluffy black fur with white paws and belly.</p>
  <img itemprop="img" src="hedral.jpeg" alt="" title="Hedral, age 18 months">
</section>
```

In this example the "https://example.org/animals#cat" item has three properties, a "name" ("Hedral"), a "desc" ("Hedral is..."), and an "img" ("hedral.jpeg").

The type gives the context for the properties, thus selecting a vocabulary: a property named "class" given for an item with the type "https://census.example/person" might refer to the economic class of an individual, while a property named "class" given for an item with the type "https://example.com/school/teacher" might refer to the classroom a teacher has been assigned. Several types can share a vocabulary. For example, the types "https://example.org/people/teacher" and "https://example.org/people/engineer" could be defined to use the same vocabulary (though maybe some properties would not be especially useful in both cases, e.g. maybe the "https://example.org/people/engineer" type might not typically be used with the "classroom" property). Multiple types defined to use the same vocabulary can be given for a single item by listing the URLs as a space-separated list in the attribute value. An item cannot be given two types if they do not use the same vocabulary, however.
5.1.4 Global identifiers for items

This section is non-normative.

Sometimes, an item gives information about a topic that has a global identifier. For example, books can be identified by their ISBN number.

Vocabularies (as identified by the `itemtype` attribute) can be designed such that items get associated with their global identifier in an unambiguous way by expressing the global identifiers as URLs given in an `itemid` attribute. The exact meaning of the URLs given in `itemid` attributes depends on the vocabulary used.

Example

Here, an item is talking about a particular book:

```html
<dl itemscope itemtype="https://vocab.example.net/book"
    itemid="urn:isbn:0-330-34032-8">
  <dt>Title</dt>
  <dd itemprop="title">The Reality Dysfunction</dd>
  <dt>Author</dt>
  <dd itemprop="author">Peter F. Hamilton</dd>
  <dt>Publication date</dt>
  <dd><time itemprop="pubdate" datetime="1996-01-26">26 January 1996</time></dd>
</dl>
```

The "https://vocab.example.net/book" vocabulary in this example would define that the `itemid` attribute takes a `urn: URL` pointing to the ISBN of the book.

5.1.5 Selecting names when defining vocabularies

This section is non-normative.

Using microdata means using a vocabulary. For some purposes, an ad-hoc vocabulary is adequate. For others, a vocabulary will need to be designed. Where possible, authors are encouraged to re-use existing vocabularies, as this makes content re-use easier.

When designing new vocabularies, identifiers can be created either using URLs, or, for properties, as plain words (with no dots or colons). For URLs, conflicts with other vocabularies can be avoided by only using identifiers that correspond to pages that the author has control over.

Example

For instance, if Jon and Adam both write content at example.com, at https://example.com/~jon/... and https://example.com/~adam/... respectively, then they could select identifiers of the form "https://example.com/~jon/name" and "https://example.com/~adam/name" respectively.

Properties whose names are just plain words can only be used within the context of the types for which they are intended; properties named using URLs can be reused in items of any type. If an item has no type, and is not part of another item, then if its properties have names that are just plain words, they are not intended to be globally unique, and are instead only intended for limited use. Generally speaking, authors are encouraged to use either properties with globally unique names (URLs) or ensure that their items are typed.

Example

Here, an item is an "https://example.org/animals#cat", and most of the properties have names that are words defined in the context of that type. There are also a few additional properties whose names come from other vocabularies.

```html
<section itemscope itemtype="https://example.org/animals#cat">
  <h1 itemprop="name https://example.com/fn">Hedral</h1>
  <p itemprop="desc">Hedral is a male american domestic</p>
</section>
```
The microdata model consists of groups of name-value pairs known as items. Each group is known as an item. Each item can have item types, a global identifier (if the vocabulary specified by the item types support global identifiers for items), and a list of name-value pairs. Each name in the name-value pair is known as a property, and each property has one or more values. Each value is either a string or itself a group of name-value pairs (an item). The names are unordered relative to each other, but if a particular name has multiple values, they do have a relative order.

Every HTML element may have an itemprop attribute specified. The itemprop attribute is a boolean attribute.

An element with the itemprop attribute specified creates a new item, a group of name-value pairs.

Elements with an itemprop attribute may have an itemtype attribute specified, to give the itemtypes of the item.

The itemtype attribute, if specified, must have a value that is an unordered set of unique space-separated tokens, none of which are identical to another token and each of which is a valid URL string that is an absolute URL, and all of which are defined to use the same vocabulary. The attribute's value must have at least one token.

The itemtypes of an item are the tokens obtained by splitting the element's itemtype attribute's value on ASCII whitespace. If the itemtype attribute is missing or parsing it in this way finds no tokens, the item is said to have no itemtypes.

The itemtypes must all be types defined in applicable specifications and must all be defined to use the same vocabulary.

Except if otherwise specified by that specification, the URLs given as the itemtypes should not be automatically dereferenced.

### Note

A specification could define that its itemtype can be dereferenced to provide the user with help information, for example. In fact, vocabulary authors are encouraged to provide useful information at the given URL.

Itemtypes are opaque identifiers, and user agents must not dereference unknown itemtypes, or otherwise deconstruct them, in order to determine how to process items that use them.

The itemtype attribute must not be specified on elements that do not have an itemprop attribute specified.
An item is said to be a typed item when either it has an item type, or it is the value of a property of a typed item. The relevant types for a typed item is the item's item types, if it has any, or else is the relevant types of the item for which it is a property's value.

Elements with an itemscope attribute and an itemtype attribute that references a vocabulary that is defined to support global identifiers for items may also have an itemid attribute specified, to give a global identifier for the item, so that it can be related to other items on pages elsewhere on the web.

The itemid attribute, if specified, must have a value that is a valid URL potentially surrounded by spaces.

The global identifier of an item is the value of its element's itemid attribute, if it has one, parsed relative to the node document of the element on which the attribute is specified. If the itemid attribute is missing or if resolving it fails, it is said to have no global identifier.

The itemid attribute must not be specified on elements that do not have both an itemscope attribute and an itemtype attribute specified, and must not be specified on elements with an itemscope attribute whose itemtype attribute specifies a vocabulary that does not support global identifiers for items, as defined by that vocabulary's specification.

The exact meaning of a global identifier is determined by the vocabulary's specification. It is up to such specifications to define whether multiple items with the same global identifier (whether on the same page or on different pages) are allowed to exist, and what the processing rules for that vocabulary are with respect to handling the case of multiple items with the same ID.

Elements with an itemscope attribute may have an itemref attribute specified, to give a list of additional elements to crawl to find the name-value pairs of the item.

The itemref attribute, if specified, must have a value that is an unordered set of unique space-separated tokens none of which are identical to another token and consisting of IDs of elements in the same tree.

The itemref attribute must not be specified on elements that do not have an itemscope attribute specified.

The itemref attribute is not part of the microdata data model. It is merely a syntactic construct to aid authors in adding annotations to pages where the data to be annotated does not follow a convenient tree structure. For example, it allows authors to mark up data in a table so that each column defines a separate item, while keeping the properties in the cells.

This example shows a simple vocabulary used to describe the products of a model railway manufacturer. The vocabulary has just five property names:

product-code
An integer that names the product in the manufacturer's catalog.

name
A brief description of the product.

scale
One of "HO", "1", or "Z" (potentially with leading or trailing whitespace), indicating the scale of the product.

digital
If present, one of "Digital", "Delta", or "Systems" (potentially with leading or trailing whitespace) indicating that the product has a digital decoder of the given type.

track-type
For track-specific products, one of "K", "M", "C" (potentially with leading or trailing whitespace) indicating the type of track for which the product is intended.

This vocabulary has four defined item types:

https://md.example.com/locos
Rolling stock with an engine

https://md.example.com/trainsets
Train sets

https://md.example.com/locomotives
Locomotives

https://md.example.com/trains
Trains
https://md.example.com/passengers
  Passenger rolling stock

https://md.example.com/track
  Track pieces

https://md.example.com/lighting
  Equipment with lighting

Each item that uses this vocabulary can be given one or more of these types, depending on what the product is.

Thus, a locomotive might be marked up as:

```html
<dl itemscope itemtype="https://md.example.com/loco
  https://md.example.com/lighting">
  <dt>Name:
  <dd itemprop="name">Tank Locomotive (DB 80)
  <dt>Product code:
  <dd itemprop="product-code">33041
  <dt>Scale:
  <dd itemprop="scale">HO
  <dt>Digital:
  <dd itemprop="digital">Delta
</dl>
```

A turnout lantern retrofit kit might be marked up as:

```html
<dl itemscope itemtype="https://md.example.com/track
  https://md.example.com/lighting">
  <dt>Name:
  <dd itemprop="name">Turnout Lantern Kit
  <dt>Product code:
  <dd itemprop="product-code">74470
  <dt>Purpose:
  <dd>For retrofitting 2 <span itemprop="track-type">C</span> Track
  turnout sets. <meta itemprop="scale" content="HO">
</dl>
```

A passenger car with no lighting might be marked up as:

```html
<dl itemscope itemtype="https://md.example.com/passengers">
  <dt>Name:
  <dd itemprop="name">Express Train Passenger Car (DB Am 203)
  <dt>Product code:
  <dd itemprop="product-code">8710
  <dt>Scale:
  <dd itemprop="scale">Z
</dl>
```

Great care is necessary when creating new vocabularies. Often, a hierarchical approach to types can be taken that results in a vocabulary where each item only ever has a single type, which is generally much simpler to manage.

### 5.2.3 Names: the `itemprop` attribute

Every HTML element may have an `itemprop` attribute specified, if doing so adds one or more properties (as defined below).

The `itemprop` attribute, if specified, must have a value that is an unordered set of unique space-separated tokens, none of which...
are identical to another token, representing the names of the name-value pairs that it adds. The attribute's value must have at least one token.

Each token must be either:

- If the item is a typed item: a defined property name allowed in this situation according to the specification that defines the relevant types for the item, or
- A valid URL string that is an absolute URL defined as an item property name allowed in this situation by a vocabulary specification, or
- A valid URL string that is an absolute URL, used as a proprietary item property name (i.e. one used by the author for private purposes, not defined in a public specification), or
- If the item is not a typed item: a string that contains no U+002E FULL STOP characters (.) and no U+003A COLON characters (:), used as a proprietary item property name (i.e. one used by the author for private purposes, not defined in a public specification).

Specifications that introduce defined property names must ensure all such property names contain no U+002E FULL STOP characters (.), no U+003A COLON characters (:), and no ASCII whitespace.

Note: The rules above disallow U+003A COLON characters (:) in non-URL values because otherwise they could not be distinguished from URLs. Values with U+002E FULL STOP characters (.) are reserved for future extensions. ASCII whitespace are disallowed because otherwise the values would be parsed as multiple tokens.

When an element with an itemprop attribute adds a property to multiple items, the requirement above regarding the tokens applies for each item individually.

The property names of an element are the tokens that the element's itemprop attribute is found to contain when its value is split on ASCII whitespace, with the order preserved but with duplicates removed (leaving only the first occurrence of each name).

Within an item, the properties are unordered with respect to each other, except for properties with the same name, which are ordered in the order they are given by the algorithm that defines the properties of an item.

Example: In the following example, the "a" property has the values "1" and "2", in that order, but whether the "a" property comes before the "b" property or not is not important:

```html
<div itemscope>
  <p itemprop="a">1</p>
  <p itemprop="a">2</p>
  <p itemprop="b">test</p>
</div>
```

Thus, the following is equivalent:

```html
<div itemscope>
  <p itemprop="b">test</p>
  <p itemprop="a">1</p>
  <p itemprop="a">2</p>
</div>
```

As is the following:

```html
<div itemscope>
  <p itemprop="a">1</p>
  <p itemprop="b">test</p>
  <p itemprop="a">2</p>
</div>
```
The property value of a name-value pair added by an element with an `itemprop` attribute is as given for the first matching case in the following list:

→ If the element also has an `itemscope` attribute
   The value is the `item` created by the element.

→ If the element is a `meta` element
   The value is the value of the element's `content` attribute, if any, or the empty string if there is no such attribute.

→ If the element is an `audio`, `embed`, `iframe`, `img`, `source`, `track`, or `video` element
   The value is the resulting URL string that results from parsing the value of the element's `src` attribute relative to the `node document` of the element at the time the attribute is set, or the empty string if there is no such attribute or if parsing results in an error.

→ If the element is an `a`, `area`, or `link` element
   The value is the resulting URL string that results from parsing the value of the element's `href` attribute relative to the `node document` of the element at the time the attribute is set, or the empty string if there is no such attribute or if parsing results in an error.

→ If the element is an `object` element
   The value is the resulting URL string that results from parsing the value of the element's `data` attribute relative to the `node document` of the element at the time the attribute is set, or the empty string if there is no such attribute or if parsing results in an error.

→ If the element is a `data` element
   The value is the value of the element's `value` attribute, if it has one, or the empty string otherwise.

→ If the element is a `meter` element
   The value is the value of the element's `value` attribute, if it has one, or the empty string otherwise.

→ If the element is a `time` element
   The value is the element's `datetime value`.

→ Otherwise
   The value is the element's descendant text content.

The URL property elements are the `a`, `area`, `audio`, `embed`, `iframe`, `img`, `link`, `object`, `source`, `track`, and `video` elements.

If a property's `value`, as defined by the property's definition, is an `absolute URL`, the property must be specified using a URL property element.

Note

These requirements do not apply just because a property value happens to match the syntax for a URL. They only apply if the property is explicitly defined as taking such a value.
To find the properties of an item defined by the element root, the user agent must run the following steps. These steps are also used to flag microdata errors.

1. Let results, memory, and pending be empty lists of elements.
2. Add the element root to memory.
3. Add the child elements of root, if any, to pending.
4. If root has an itemref attribute, split the value of that itemref attribute on ASCII whitespace. For each resulting token ID, if there is an element in the tree of root with the ID ID, then add the first such element to pending.
5. While pending is not empty:
   1. Remove an element from pending and let current be that element.
   2. If current is already in memory, there is a microdata error; continue.
   3. Add current to memory.
   4. If current does not have an itemscope attribute, then: add all the child elements of current to pending.
   5. If current has an itemprop attribute specified and has one or more property names, then add current to results.
6. Sort results in tree order.
7. Return results.

A document must not contain any items for which the algorithm to find the properties of an item finds any microdata errors.

An item is a top-level microdata item if its element does not have an itemprop attribute.

All itemref attributes in a Document must be such that there are no cycles in the graph formed from representing each item in the Document as a node in the graph and each property of an item whose value is another item as an edge in the graph connecting those two items.

A document must not contain any elements that have an itemprop attribute that would not be found to be a property of any of the items in that document were their properties all to be determined.

Example

For example, a book about the first moon landing could be called "mission:moon". A "title" property from a vocabulary that defines a title as being a string would not expect the title to be given in an element, even though it looks like a URL. On the other hand, if there was a (rather narrowly scoped!) vocabulary for "books whose titles look like URLs" which had a "title" property defined to take a URL, then the property would expect the title to be given in an element (or one of the other URL property elements), because of the requirement above.

5.2.5 Associating names with items

To find the properties of an item defined by the element root, the user agent must run the following steps. These steps are also used to flag microdata errors.

Example

In this example, a single license statement is applied to two works, using itemref from the items representing the works:

```html
<!DOCTYPE HTML>
<html lang="en">
<head>
  <title>Photo gallery</title>
</head>
<body>
  <h1>My photos</h1>
  <figure itemscope itemtype="http://n.whatwg.org/work" itemref="licenses">
    <img itemprop="work" src="images/house.jpeg" alt="A white house, boarded up, sits in a forest.">
    <figcaption itemprop="title">The house I found.</figcaption>
  </figure>
</body>
</html>
```
Currently, the `itemscope`, `itemprop`, and other microdata attributes are only defined for HTML elements. This means that attributes with the literal names "itemscope", "itemprop", etc, do not cause microdata processing to occur on elements in other namespaces, such as SVG.

The above results in two items with the type "http://n.whatwg.org/work", one with:

- **work**: images/house.jpeg
- **title**: The house I found.
- **license**: http://www.opensource.org/licenses/mit-license.php

...and one with:

- **work**: images/mailbox.jpeg
- **title**: The mailbox.
- **license**: http://www.opensource.org/licenses/mit-license.php

Thus, in the following example there is only one item, not two.

```
<p itemscope></p> <!-- this is an item (with no properties and no type) -->
<svg itemscope></svg> <!-- this is not, it's just an SVG svg element with an invalid unknown attribute -->
```

### 5.2.6 Microdata and other namespaces

Currently, the `itemscope`, `itemprop`, and other microdata attributes are only defined for HTML elements. This means that attributes with the literal names "itemscope", "itemprop", etc, do not cause microdata processing to occur on elements in other namespaces, such as SVG.

**Example**

Thus, in the following example there is only one item, not two.

```
<p itemscope></p> <!-- this is an item (with no properties and no type) -->
<svg itemscope></svg> <!-- this is not, it's just an SVG svg element with an invalid unknown attribute -->
```

### 5.3 Sample microdata vocabularies

The vocabularies in this section are primarily intended to demonstrate how a vocabulary is specified, though they are also usable in their own right.
5.3.1 vCard

An item with the item type \texttt{http://microformats.org/profile/hcard} represents a person's or organization's contact information.

This vocabulary does not support global identifiers for items.

The following are the type's defined property names. They are based on the vocabulary defined in vCard Format Specification (vCard) and its extensions, where more information on how to interpret the values can be found. \[\text{RFC6350}\]

\textbf{kind}

Describes what kind of contact the item represents.

The value must be text that is identical to one of the kind strings.

A single property with the name \texttt{kind} may be present within each item with the type \texttt{http://microformats.org/profile/hcard}.

\textbf{fn}

Gives the formatted text corresponding to the name of the person or organization.

The value must be text.

Exactly one property with the name \texttt{fn} must be present within each item with the type \texttt{http://microformats.org/profile/hcard}.

\textbf{n}

Gives the structured name of the person or organization.

The value must be an item with zero or more of each of the family-name, given-name, additional-name, honorific-prefix, and honorific-suffix properties.

Exactly one property with the name \texttt{n} must be present within each item with the type \texttt{http://microformats.org/profile/hcard}.

\textbf{family-name (inside \texttt{n})}

Gives the family name of the person, or the full name of the organization.

The value must be text.

Any number of properties with the name family-name may be present within the item that forms the value of the n property of an item with the type \texttt{http://microformats.org/profile/hcard}.

\textbf{given-name (inside \texttt{n})}

Gives the given-name of the person.

The value must be text.

Any number of properties with the name given-name may be present within the item that forms the value of the n property of an item with the type \texttt{http://microformats.org/profile/hcard}.

\textbf{additional-name (inside \texttt{n})}

Gives the any additional names of the person.

The value must be text.

Any number of properties with the name additional-name may be present within the item that forms the value of the n property of an item with the type \texttt{http://microformats.org/profile/hcard}.

\textbf{honorific-prefix (inside \texttt{n})}

Gives the honorific prefix of the person.

The value must be text.

Any number of properties with the name honorific-prefix may be present within the item that forms the value of the n property of an item with the type \texttt{http://microformats.org/profile/hcard}.  

743
n⁷⁴³ property of an item⁷³⁶ with the type http://microformats.org/profile/hcard⁷⁴³.

**honorific-suffix (inside n⁷⁴³)**

Gives the honorific suffix of the person.

The value⁷⁴⁰ must be text.

Any number of properties with the name honorific-suffix⁷⁴⁴ may be present within the item⁷³⁶ that forms the value⁷⁴⁰ of the n⁷⁴³ property of an item⁷³⁶ with the type http://microformats.org/profile/hcard⁷⁴³.

**nickname**

Gives the nickname of the person or organization.

The value⁷⁴⁰ must be text.

Any number of properties with the name nickname⁷⁴⁴ may be present within each item⁷³⁶ with the type http://microformats.org/profile/hcard⁷⁴³.

**photo**

Gives a photograph of the person or organization.

The value⁷⁴⁰ must be an absolute URL.

Any number of properties with the name photo⁷⁴⁴ may be present within each item⁷³⁶ with the type http://microformats.org/profile/hcard⁷⁴³.

**bday**

Gives the birth date of the person or organization.

The value⁷⁴⁰ must be a valid date string⁷⁵.

A single property with the name bday⁷⁴⁴ may be present within each item⁷³⁶ with the type http://microformats.org/profile/hcard⁷⁴³.

**anniversary**

Gives the birth date of the person or organization.

The value⁷⁴⁰ must be a valid date string⁷⁵.

A single property with the name anniversary⁷⁴⁴ may be present within each item⁷³⁶ with the type http://microformats.org/profile/hcard⁷⁴³.

**sex**

Gives the biological sex of the person.

The value⁷⁴⁰ must be one of F, meaning “female”, M, meaning “male”, N, meaning “none or not applicable”, 0, meaning “other”, or U, meaning “unknown”.

A single property with the name sex⁷⁴⁴ may be present within each item⁷³⁶ with the type http://microformats.org/profile/hcard⁷⁴³.

**gender-identity**

Gives the gender identity of the person.

The value⁷⁴⁰ must be text.

A single property with the name gender-identity⁷⁴⁴ may be present within each item⁷³⁶ with the type http://microformats.org/profile/hcard⁷⁴³.
adr
Gives the delivery address of the person or organization.

The value must be an item with zero or more type, post-office-box, extended-address, and street-address properties, and optionally a locality property, optionally a region property, optionally a postal-code property, and optionally a country-name property.

If no type properties are present within an item that forms the value of an adr property of an item with the type http://microformats.org/profile/hcard, then the address type string work is implied.

Any number of properties with the name adr may be present within each item with the type http://microformats.org/profile/hcard.

type (inside adr)
Gives the type of delivery address.

The value must be text that is identical to one of the address type strings.

Any number of properties with the name type may be present within the item that forms the value of an adr property of an item with the type http://microformats.org/profile/hcard, but within each such adr property there must only be one type property per distinct value.

post-office-box (inside adr)
Gives the post office box component of the delivery address of the person or organization.

The value must be text.

Any number of properties with the name post-office-box may be present within the item that forms the value of an adr property of an item with the type http://microformats.org/profile/hcard.

Note vCard urges authors not to use this field.

extended-address (inside adr)
Gives an additional component of the delivery address of the person or organization.

The value must be text.

Any number of properties with the name extended-address may be present within the item that forms the value of an adr property of an item with the type http://microformats.org/profile/hcard.

Note vCard urges authors not to use this field.

street-address (inside adr)
Gives the street address component of the delivery address of the person or organization.

The value must be text.

Any number of properties with the name street-address may be present within the item that forms the value of an adr property of an item with the type http://microformats.org/profile/hcard.

locality (inside adr)
Gives the locality component (e.g. city) of the delivery address of the person or organization.

The value must be text.

A single property with the name locality may be present within the item that forms the value of an adr property of an item with the type http://microformats.org/profile/hcard.

region (inside adr)
Gives the region component (e.g. state or province) of the delivery address of the person or organization.
The value must be text.

A single property with the name `region` may be present within the item that forms the value of an `adr` property of an item, with the type `http://microformats.org/profile/hcard`.

**postal-code (inside adr)**

Gives the postal code component of the delivery address of the person or organization.

The value must be text.

A single property with the name `postal-code` may be present within the item that forms the value of an `adr` property of an item, with the type `http://microformats.org/profile/hcard`.

**country-name (inside adr)**

Gives the country name component of the delivery address of the person or organization.

The value must be text.

A single property with the name `country-name` may be present within the item that forms the value of an `adr` property of an item, with the type `http://microformats.org/profile/hcard`.

**tel**

Gives the telephone number of the person or organization.

The value must be either text that can be interpreted as a telephone number as defined in the CCITT specifications E.163 and X.121, or an item with zero or more `type` properties and exactly one `value` property. [E163][1272] [X121][1278]

If no `type` properties are present within an item that forms the value of a `tel` property of an item with the type `http://microformats.org/profile/hcard`, or if the value of such a `tel` property is text, then the telephone type string `voice` is implied.

Any number of properties with the name `tel` may be present within each item with the type `http://microformats.org/profile/hcard`.

**type (inside tel)**

Gives the type of telephone number.

The value must be text that is identical to one of the telephone type strings.

Any number of properties with the name `type` may be present within the item that forms the value of a `tel` property of an item, with the type `http://microformats.org/profile/hcard`, but within each such `tel` property item there must only be one `type` property per distinct value.

**value (inside tel)**

Gives the actual telephone number of the person or organization.

The value must be text that can be interpreted as a telephone number as defined in the CCITT specifications E.163 and X.121. [E163][1272] [X121][1278]

Exactly one property with the name `value` must be present within the item that forms the value of a `tel` property of an item, with the type `http://microformats.org/profile/hcard`.

**email**

Gives the email address of the person or organization.

The value must be text.

Any number of properties with the name `email` may be present within each item with the type `http://microformats.org/profile/hcard`.

**impp**

Gives a URL for instant messaging and presence protocol communications with the person or organization.

The value must be an absolute URL.
Any number of properties with the name `impp` may be present within each `item` with the type `http://microformats.org/profile/hcard`.

**lang**

Gives a language understood by the person or organization.

The `value` must be a valid BCP 47 language tag. [BCP47]

Any number of properties with the name `lang` may be present within each `item` with the type `http://microformats.org/profile/hcard`.

**tz**

Gives the time zone of the person or organization.

The `value` must be text and must match the following syntax:

1. Either a U+002B PLUS SIGN character (+) or a U+002D HYPHEN-MINUS character (-).
2. A `valid non-negative integer` that is exactly two digits long and that represents a number in the range 00..23.
3. A U+003A COLON character (:).
4. A `valid non-negative integer` that is exactly two digits long and that represents a number in the range 00..59.

Any number of properties with the name `tz` may be present within each `item` with the type `http://microformats.org/profile/hcard`.

**geo**

Gives the geographical position of the person or organization.

The `value` must be text and must match the following syntax:

1. Optionally, either a U+002B PLUS SIGN character (+) or a U+002D HYPHEN-MINUS character (-).
2. One or more ASCII digits.
3. Optionally*, a U+002E FULL STOP character (.) followed by one or more ASCII digits.
4. A U+003B SEMICOLON character (;).
5. Optionally, either a U+002B PLUS SIGN character (+) or a U+002D HYPHEN-MINUS character (-).
6. One or more ASCII digits.
7. Optionally*, a U+002E FULL STOP character (.) followed by one or more ASCII digits.

The optional components marked with an asterisk (*) should be included, and should have six digits each.

The value specifies latitude and longitude, in that order (i.e., "LAT LON" ordering), in decimal degrees. The longitude represents the location east and west of the prime meridian as a positive or negative real number, respectively. The latitude represents the location north and south of the equator as a positive or negative real number, respectively.

Any number of properties with the name `geo` may be present within each `item` with the type `http://microformats.org/profile/hcard`.

**title**

Gives the job title, functional position or function of the person or organization.

The `value` must be text.

Any number of properties with the name `title` may be present within each `item` with the type `http://microformats.org/profile/hcard`.

**role**

Gives the role, occupation, or business category of the person or organization.
The value must be text.

Any number of properties with the name role may be present within each item with the type http://microformats.org/profile/hcard.

**logo**

Gives the logo of the person or organization.

The value must be an absolute URL.

Any number of properties with the name logo may be present within each item with the type http://microformats.org/profile/hcard.

**agent**

Gives the contact information of another person who will act on behalf of the person or organization.

The value must be either an item with the type http://microformats.org/profile/hcard, or an absolute URL, or text.

Any number of properties with the name agent may be present within each item with the type http://microformats.org/profile/hcard.

**org**

Gives the name and units of the organization.

The value must be either text or an item with one organization-name property and zero or more organization-unit properties.

Any number of properties with the name org may be present within each item with the type http://microformats.org/profile/hcard.

**organization-name (inside org)**

Gives the name of the organization.

The value must be text.

Exactly one property with the name organization-name must be present within the item that forms the value of an org property of an item with the type http://microformats.org/profile/hcard.

**organization-unit (inside org)**

Gives the name of the organization unit.

The value must be text.

Any number of properties with the name organization-unit may be present within the item that forms the value of the org property of an item with the type http://microformats.org/profile/hcard.

**member**

Gives a URL that represents a member of the group.

The value must be an absolute URL.

Any number of properties with the name member may be present within each item with the type http://microformats.org/profile/hcard if the item also has a property with the name kind whose value is "group".

**related**

Gives a relationship to another entity.

The value must be an item with one url property and one rel properties.

Any number of properties with the name related may be present within each item with the type http://microformats.org/profile/hcard.

**url (inside related)**

Gives the URL for the related entity.
The value must be an absolute URL.

Exactly one property with the name url must be present within the item that forms the value of a related property of an item with the type http://microformats.org/profile/hcard.

rel (inside related)

Gives the relationship between the entity and the related entity.

The value must be text that is identical to one of the relationship strings.

Exactly one property with the name rel must be present within the item that forms the value of a related property of an item with the type http://microformats.org/profile/hcard.

categories

Gives the name of a category or tag that the person or organization could be classified as.

The value must be text.

Any number of properties with the name categories may be present within each item with the type http://microformats.org/profile/hcard.

note

Gives supplemental information or a comment about the person or organization.

The value must be text.

Any number of properties with the name note may be present within each item with the type http://microformats.org/profile/hcard.

rev

Gives the revision date and time of the contact information.

The value must be text that is a valid global date and time string.

Note

The value distinguishes the current revision of the information for other renditions of the information.

Any number of properties with the name rev may be present within each item with the type http://microformats.org/profile/hcard.

sound

Gives a sound file relating to the person or organization.

The value must be an absolute URL.

Any number of properties with the name sound may be present within each item with the type http://microformats.org/profile/hcard.

uid

Gives a globally unique identifier corresponding to the person or organization.

The value must be text.

A single property with the name uid may be present within each item with the type http://microformats.org/profile/hcard.

url

Gives a URL relating to the person or organization.

The value must be an absolute URL.

Any number of properties with the name url may be present within each item with the type http://microformats.org/profile/hcard.
The **kind strings** are:

- **individual**
  - Indicates a single entity (e.g. a person).

- **group**
  - Indicates multiple entities (e.g. a mailing list).

- **org**
  - Indicates a single entity that is not a person (e.g. a company).

- **location**
  - Indicates a geographical place (e.g. an office building).

The **address type strings** are:

- **home**
  - Indicates a delivery address for a residence.

- **work**
  - Indicates a delivery address for a place of work.

The **telephone type strings** are:

- **home**
  - Indicates a residential number.

- **work**
  - Indicates a telephone number for a place of work.

- **text**
  - Indicates that the telephone number supports text messages (SMS).

- **voice**
  - Indicates a voice telephone number.

- **fax**
  - Indicates a facsimile telephone number.

- **cell**
  - Indicates a cellular telephone number.

- **video**
  - Indicates a video conferencing telephone number.

- **pager**
  - Indicates a paging device telephone number.

- **textphone**
  - Indicates a telecommunication device for people with hearing or speech difficulties.

The **relationship strings** are:

- **emergency**
  - An emergency contact.

- **agent**
  - Another entity that acts on behalf of this entity.
5.3.1.1 Conversion to vCard

Given a list of nodes $nodes$ in a Document, a user agent must run the following algorithm to extract any vCard data represented by those nodes (only the first vCard is returned):

1. If none of the nodes in $nodes$ are items with the item type http://microformats.org/profile/hcard, then there is no vCard. Abort the algorithm, returning nothing.

2. Let $node$ be the first node in $nodes$ that is an item with the item type http://microformats.org/profile/hcard.

3. Let $output$ be an empty string.

4. Add a vCard line with the type "BEGIN" and the value "VCARD" to $output$.

5. Add a vCard line with the type "PROFILE" and the value "VCARD" to $output$.

6. Add a vCard line with the type "VERSION" and the value "4.0" to $output$.

7. Add a vCard line with the type "SOURCE" and the result of escaping the vCard text string that is the document's URL as the value to $output$.

8. If the title element is not null, add a vCard line with the type "NAME" and with the result of escaping the vCard text string obtained from the title element's descendant text content as the value to $output$.

9. Let $sex$ be the empty string.

10. Let gender-identity be the empty string.

11. For each element $element$ that is a property of the item node: for each name $name$ in $element$'s property names, run the following substeps:

   1. Let parameters be an empty set of name-value pairs.

   2. Run the appropriate set of substeps from the following list. The steps will set a variable $value$, which is used in the next step.

      If the property's value is an item subitem and name is

      1. Let $value$ be the empty string.

      2. Append to $value$ the result of collecting the first vCard subproperty named family-name in subitem.
3. Append a U+003B SEMICOLON character (;) to value.

4. Append to value the result of collecting the first vCard subproperty named given-name in subitem.

5. Append a U+003B SEMICOLON character (;) to value.

6. Append to value the result of collecting the first vCard subproperty named additional-name in subitem.

7. Append a U+003B SEMICOLON character (;) to value.

8. Append to value the result of collecting the first vCard subproperty named honorific-prefix in subitem.

9. Append a U+003B SEMICOLON character (;) to value.

10. Append to value the result of collecting the first vCard subproperty named honorific-suffix in subitem.

If the property's value is an item subitem and name is adr

1. Let value be the empty string.

2. Append to value the result of collecting vCard subproperties named post-office-box in subitem.

3. Append a U+003B SEMICOLON character (;) to value.

4. Append to value the result of collecting vCard subproperties named extended-address in subitem.

5. Append a U+003B SEMICOLON character (;) to value.

6. Append to value the result of collecting vCard subproperties named street-address in subitem.

7. Append a U+003B SEMICOLON character (;) to value.

8. Append to value the result of collecting the first vCard subproperty named locality in subitem.

9. Append a U+003B SEMICOLON character (;) to value.

10. Append to value the result of collecting the first vCard subproperty named region in subitem.

11. Append a U+003B SEMICOLON character (;) to value.

12. Append to value the result of collecting the first vCard subproperty named postal-code in subitem.

13. Append a U+003B SEMICOLON character (;) to value.

14. Append to value the result of collecting the first vCard subproperty named country-name in subitem.

15. If there is a property named type in subitem, and the first such property has a value that is not an item, whose value consists only of ASCII alphanumerics, then add a parameter named "TYPE" whose value is the value of that property to parameters.

If the property's value is an item subitem and name is org

1. Let value be the empty string.

2. Append to value the result of collecting the first vCard subproperty named organization-name in subitem.

3. For each property named organization-unit in subitem, run the following steps:

   1. If the value of the property is an item, then skip this property.
2. Append a U+003B SEMICOLON character (;) to value.

3. Append the result of escaping the vCard text string by the value of the property to value.

If the property's value is an item subitem with the item type and name is related

1. Let value be the empty string.

2. If there is a property named url in subitem, and its element is a URL property element, then append the result of escaping the vCard text string by the value of the first such property to value, and add a parameter with the name "VALUE" and the value "URI" to parameters.

3. If there is a property named rel in subitem, and the first such property has a value that is not an item and whose value consists only of ASCII alphanumeric, then add a parameter named "RELATION" whose value is the value of that property to parameters.

If the property's value is an item and name is none of the above

1. Let value be the result of collecting the first vCard subproperty named value in subitem.

2. If there is a property named type in subitem, and the first such property has a value that is not an item and whose value consists only of ASCII alphanumeric, then add a parameter named "TYPE" whose value is the value of that property to parameters.

If the property's value is not an item and its name is sex

If this is the first such property to be found, set sex to the property's value.

If the property's value is not an item and its name is gender-identity

If this is the first such property to be found, set gender-identity to the property's value.

Otherwise (the property's value is not an item)

1. Let value be the property's value.

2. If element is one of the URL property elements, add a parameter with the name "VALUE" and the value "URI" to parameters.

3. Otherwise, if name is bday or anniversary and the value is a valid date string, add a parameter with the name "VALUE" and the value "DATE" to parameters.

4. Otherwise, if name is rev and the value is a valid global date and time string, add a parameter with the name "VALUE" and the value "DATE-TIME" to parameters.

5. Prefix every U+005C REVERSE SOLIDUS character (\) in value with another U+005C REVERSE SOLIDUS character (\).

6. Prefix every U+002C COMMA character (,) in value with a U+005C REVERSE SOLIDUS character (\).

7. Unless name is geo, prefix every U+003B SEMICOLON character (;) in value with a U+005C REVERSE SOLIDUS character (\).

8. Replace every U+000D CARRIAGE RETURN U+000A LINE FEED character pair (CRLF) in value with a U+005C REVERSE SOLIDUS character (\) followed by a U+006E LATIN SMALL LETTER N character (n).

9. Replace every remaining U+000D CARRIAGE RETURN (CR) or U+000A LINE FEED (LF) character in value with a U+005C REVERSE SOLIDUS character (\) followed by a U+006E LATIN SMALL LETTER N character (n).

3. Add a vCard line with the type name, the parameters parameters, and the value value to output.

12. If either sex or gender-identity has a value that is not the empty string, add a vCard line with the type "GENDER" and the value consisting of the concatenation of sex, a U+003B SEMICOLON character (;), and gender-identity to output.

13. Add a vCard line with the type "END" and the value "VCARD" to output.

When the above algorithm says that the user agent is to add a vCard line consisting of a type type, optionally some parameters, and a value value to a string output, it must run the following steps:
1. Let \textit{line} be an empty string.

2. Append \textit{type}, converted to ASCII uppercase, to \textit{line}.

3. If there are any parameters, then for each parameter, in the order that they were added, run these substeps:
   1. Append a U+003B SEMICOLON character (:) to \textit{line}.
   2. Append the parameter's name to \textit{line}.
   3. Append a U+003D EQUALS SIGN character (=) to \textit{line}.
   4. Append the parameter's value to \textit{line}.

4. Append a U+003A COLON character (:) to \textit{line}.

5. Append \textit{value} to \textit{line}.

6. Let \textit{maximum length} be 75.

7. While \textit{line}'s code point length is greater than \textit{maximum length}:
   1. Append the first \textit{maximum length} code points of \textit{line} to \textit{output}.
   2. Remove the first \textit{maximum length} code points from \textit{line}.
   3. Append a U+000D CARRIAGE RETURN character (CR) to \textit{output}.
   4. Append a U+000A LINE FEED character (LF) to \textit{output}.
   5. Append a U+0020 SPACE character to \textit{output}.
   6. Let \textit{maximum length} be 74.

8. Append (what remains of) \textit{line} to \textit{output}.

9. Append a U+000D CARRIAGE RETURN character (CR) to \textit{output}.

10. Append a U+000A LINE FEED character (LF) to \textit{output}.

When the steps above require the user agent to obtain the result of collecting \textbf{vCard subproperties} named \textit{subname} in \textit{subitem}, the user agent must run the following steps:

1. Let \textit{value} be the empty string.

2. For each property named \textit{subname} in the item \textit{subitem}, run the following substeps:
   1. If the \textit{value} of the property is itself an \textit{item}, then skip this property.
   2. If this is not the first property named \textit{subname} in \textit{subitem} (ignoring any that were skipped by the previous step), then append a U+002C COMMA character (,) to \textit{value}.
   3. Append the result of \textbf{escaping the vCard text string} given by the \textit{value} of the property to \textit{value}.

3. Return \textit{value}.

When the steps above require the user agent to obtain the result of collecting the first \textbf{vCard subproperty} named \textit{subname} in \textit{subitem}, the user agent must run the following steps:

1. If there are no properties named \textit{subname} in \textit{subitem}, then return the empty string.

2. If the \textit{value} of the first property named \textit{subname} in \textit{subitem} is an \textit{item}, then return the empty string.

3. Return the result of \textbf{escaping the vCard text string} given by the \textit{value} of the first property named \textit{subname} in \textit{subitem}.

When the above algorithms say the user agent is to \textbf{escape the vCard text string} \textit{value}, the user agent must use the following steps:

1. Prefix every U+005C REVERSE SOLIDUS character (\) in \textit{value} with another U+005C REVERSE SOLIDUS character (\).

2. Prefix every U+002C COMMA character (,) in \textit{value} with a U+005C REVERSE SOLIDUS character (\).
3. Prefix every U+003B SEMICOLON character (;) in value with a U+005C REVERSE SOLIDUS character (\).

4. Replace every U+000D CARRIAGE RETURN U+000A LINE FEED character pair (CRLF) in value with a U+005C REVERSE SOLIDUS character (\) followed by a U+006E LATIN SMALL LETTER N character (n).

5. Replace every remaining U+000D CARRIAGE RETURN (CR) or U+000A LINE FEED (LF) character in value with a U+005C REVERSE SOLIDUS character (\) followed by a U+006E LATIN SMALL LETTER N character (n).

6. Return the mutated value.

Note
This algorithm can generate invalid vCard output, if the input does not conform to the rules described for the http://microformats.org/profile/hcard and defined property names.

5.3.1.2 Examples
This section is non-normative.

Here is a long example vCard for a fictional character called "Jack Bauer":

```html
<section id="jack" itemscope itemtype="http://microformats.org/profile/hcard">
  <h1 itemprop="fn">
    <span itemprop="n" itemscope>
      <span itemprop="given-name">Jack</span>
      <span itemprop="family-name">Bauer</span>
    </span>
  </h1>
  <img itemprop="photo" alt="" src="jack-bauer.jpg">
  <p itemprop="org">
    <span itemprop="organization-name">Counter-Terrorist Unit</span>
    (Los Angeles Division)
  </p>
  <p>
    <span itemprop="adr">
      <span itemprop="street-address">10201 W. Pico Blvd.</span>
      <span itemprop="locality">Los Angeles</span>,
      <span itemprop="region">CA</span>
      <span itemprop="postal-code">90064</span>
    </span>
    <span itemprop="country-name">United States</span>
  </p>
  <span itemprop="geo">34.052339;-118.410623</span>
</section>

<h2>Assorted Contact Methods</h2>
<ul>
  <li itemprop="tel"><span itemprop="value">+1 (310) 597 3781</span><meta itemprop="type" content="voice"></li>
  <li itemprop="tel"><span itemprop="value">+1 (310) 555 3781</span><meta itemprop="type" content="cell"></li>
  <li itemprop="email"><a href="mailto:j.bauer@la.ctu.gov.invalid">j.bauer@la.ctu.gov.invalid</a></li>
  <li itemprop="url"><a href="https://en.wikipedia.org/wiki/Jack_Bauer">I'm on Wikipedia</a> so you can leave a message on my user talk page.</li>
</ul>

<ins datetime="2008-07-20 21:00:00+01:00">
  Example
</ins>
An item with the item type `p736` http://microformats.org/profile/hcalendar#vevent represents an event. This vocabulary does not support global identifiers for items.

The following are the type's defined property names. They are based on the vocabulary defined in `Internet Calendaring and Scheduling Core Object Specification (iCalendar)`, where more information on how to interpret the values can be found. [RFC5545]

Example

This example shows a site's contact details (using the `address` element) containing an address with two street components:

```html
<address itemscope itemtype="http://microformats.org/profile/hcard">
  <strong itemprop="fn"><span itemprop="n"><span itemprop="given-name">Alfred</span><span itemprop="family-name">Person</span></span></strong> <br>
  <span itemprop="street-address">1600 Amphitheatre Parkway</span> <br>
  <span itemprop="street-address">Building 43, Second Floor</span> <br>
  <span itemprop="locality">Mountain View</span>, <span itemprop="region">CA</span> <span itemprop="postal-code">94043</span>
</address>
```

Example

The vCard vocabulary can be used to just mark up people's names:

```html
<span itemscope itemtype="http://microformats.org/profile/hcard">
  <span itemprop="fn"><span itemprop="n"><span itemprop="given-name">George</span><span itemprop="family-name">Washington</span></span></span>
</span>
```

This creates a single item with a two name-value pairs, one with the name "fn" and the value "George Washington", and the other with the name "n" and a second item as its value, the second item having the two name-value pairs "given-name" and "family-name" with the values "George" and "Washington" respectively. This is defined to map to the following vCard:

```
BEGIN:VCARD
PROFILE:VCARD
VERSION:4.0
SOURCE:document's address
FN:George Washington
N:Washington;George;;
END:VCARD
```

5.3.2 vEvent

An item with the item type `p736` http://microformats.org/profile/hcalendar#vevent represents an event.

This vocabulary does not support global identifiers for items.

The following are the type's defined property names. They are based on the vocabulary defined in `Internet Calendaring and Scheduling Core Object Specification (iCalendar)`, where more information on how to interpret the values can be found. [RFC5545]

Note

Only the parts of the iCalendar vocabulary relating to events are used here; this vocabulary cannot express a complete iCalendar...
attach

Gives the address of an associated document for the event.

The value must be an absolute URL.

Any number of properties with the name attach may be present within each item with the type http://microformats.org/profile/hcalendar#vevent.

categories

Gives the name of a category or tag that the event could be classified as.

The value must be text.

Any number of properties with the name categories may be present within each item with the type http://microformats.org/profile/hcalendar#vevent.

class

Gives the access classification of the information regarding the event.

The value must be text with one of the following values:

• public
• private
• confidential

⚠Warning!

This is merely advisory and cannot be considered a confidentiality measure.

A single property with the name class may be present within each item with the type http://microformats.org/profile/hcalendar#vevent.

comment

Gives a comment regarding the event.

The value must be text.

Any number of properties with the name comment may be present within each item with the type http://microformats.org/profile/hcalendar#vevent.

description

Gives a detailed description of the event.

The value must be text.

A single property with the name description may be present within each item with the type http://microformats.org/profile/hcalendar#vevent.

geo

Gives the geographical position of the event.

The value must be text and must match the following syntax:

1. Optionally, either a U+002B PLUS SIGN character (+) or a U+002D HYPHEN-MINUS character (-).
2. One or more ASCII digits.
3. Optionally*, a U+002E FULL STOP character (.) followed by one or more ASCII digits.
4. A U+003B SEMICOLON character (;).
5. Optionally, either a U+002B PLUS SIGN character (+) or a U+002D HYPHEN-MINUS character (-).
6. One or more ASCII digits.

7. Optionally*, a U+002E FULL STOP character (.) followed by one or more ASCII digits.

The optional components marked with an asterisk (*) should be included, and should have six digits each.

---

**Note**

The value specifies latitude and longitude, in that order (i.e., "LAT LON" ordering), in decimal degrees. The longitude represents the location east and west of the prime meridian as a positive or negative real number, respectively. The latitude represents the location north and south of the equator as a positive or negative real number, respectively.

---

A single property with the name `geo` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

**location**

Gives the location of the event.

The value must be text.

A single property with the name `location` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

**resources**

Gives a resource that will be needed for the event.

The value must be text.

Any number of properties with the name `resources` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

**status**

Gives the confirmation status of the event.

The value must be text with one of the following values:

- tentative
- confirmed
- cancelled

A single property with the name `status` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

**summary**

Gives a short summary of the event.

The value must be text.

User agents should replace U+000A LINE FEED (LF) characters in the value by U+0020 SPACE characters when using the value.

A single property with the name `summary` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

**dtend**

Gives the date and time by which the event ends.

If the property with the name `dtend` is present within an item with the type `http://microformats.org/profile/hcalendar#vevent` that has a property with the name `dtstart` whose value is a valid date string, then the value of the property with the name `dtend` must be text that is a valid date string also. Otherwise, the value of the property must be text that is a valid global date and time string.

In either case, the value be later in time than the value of the `dtstart` property of the same item.

---

**Note**

The time given by the `dtend` property is not inclusive. For day-long events, therefore, the `dtend` property's value will...
A single property with the name \texttt{dtend} may be present within each \texttt{item} with the type \url{http://microformats.org/profile/hcalendar#vevent}, so long as that \url{http://microformats.org/profile/hcalendar#vevent} does not have a property with the name \texttt{duration}.

\textbf{dtstart}

Gives the date and time at which the event starts.

The \texttt{value} must be text that is either a \texttt{valid date string} or a \texttt{valid global date and time string}.

Exactly one property with the name \texttt{dtstart} must be present within each \texttt{item} with the type \url{http://microformats.org/profile/hcalendar#vevent}.

\textbf{duration}

Gives the duration of the event.

The \texttt{value} must be text that is a \texttt{valid vevent duration string}.

The duration represented is the sum of all the durations represented by integers in the value.

A single property with the name \texttt{duration} may be present within each \texttt{item} with the type \url{http://microformats.org/profile/hcalendar#vevent}, so long as that \url{http://microformats.org/profile/hcalendar#vevent} does not have a property with the name \texttt{dtend}.

\textbf{transp}

Gives whether the event is to be considered as consuming time on a calendar, for the purpose of free-busy time searches.

The \texttt{value} must be text with one of the following values:

- opaque
- transparent

A single property with the name \texttt{transp} may be present within each \texttt{item} with the type \url{http://microformats.org/profile/hcalendar#vevent}.

\textbf{contact}

Gives the contact information for the event.

The \texttt{value} must be text.

Any number of properties with the name \texttt{contact} may be present within each \texttt{item} with the type \url{http://microformats.org/profile/hcalendar#vevent}.

\textbf{url}

Gives a \texttt{URL} for the event.

The \texttt{value} must be an \texttt{absolute URL}.

A single property with the name \texttt{url} may be present within each \texttt{item} with the type \url{http://microformats.org/profile/hcalendar#vevent}.

\textbf{uid}

Gives a globally unique identifier corresponding to the event.

The \texttt{value} must be text.

A single property with the name \texttt{uid} may be present within each \texttt{item} with the type \url{http://microformats.org/profile/hcalendar#vevent}.

\textbf{exdate}

Gives a date and time at which the event does not occur despite the recurrence rules.
The value must be text that is either a valid date string or a valid global date and time string.

Any number of properties with the name `exdate` may be present within each item with the type http://microformats.org/profile/hcalendar#vevent.

**rdate**

Gives a date and time at which the event recurs.

The value must be text that is one of the following:

- A valid date string.
- A valid global date and time string.
- A valid global date and time string followed by a U+002F SOLIDUS character (/) followed by a second valid global date and time string representing a later time.
- A valid global date and time string followed by a U+002F SOLIDUS character (/) followed by a valid vevent duration string.

Any number of properties with the name `rdate` may be present within each item with the type http://microformats.org/profile/hcalendar#vevent.

**rrule**

Gives a rule for finding dates and times at which the event occurs.

The value must be text that matches the RECUR value type defined in iCalendar. [RFC5545]

A single property with the name `rrule` may be present within each item with the type http://microformats.org/profile/hcalendar#vevent.

**created**

Gives the date and time at which the event information was first created in a calendaring system.

The value must be text that is a valid global date and time string.

A single property with the name `created` may be present within each item with the type http://microformats.org/profile/hcalendar#vevent.

**last-modified**

Gives the date and time at which the event information was last modified in a calendaring system.

The value must be text that is a valid global date and time string.

A single property with the name `last-modified` may be present within each item with the type http://microformats.org/profile/hcalendar#vevent.

**sequence**

Gives a revision number for the event information.

The value must be text that is a valid non-negative integer.

A single property with the name `sequence` may be present within each item with the type http://microformats.org/profile/hcalendar#vevent.

A string is a valid vevent duration string if it matches the following pattern:

1. A U+0050 LATIN CAPITAL LETTER P character (P).
2. One of the following:
   - A valid non-negative integer followed by a U+0057 LATIN CAPITAL LETTER W character (W). The integer represents a duration of that number of weeks.
   - At least one, and possible both in this order, of the following:
     1. A valid non-negative integer followed by a U+0044 LATIN CAPITAL LETTER D character (D). The integer represents a duration of that number of days.
2. A U+0054 LATIN CAPITAL LETTER T character (T) followed by any one of the following, or the first and second of the following in that order, or the second and third of the following in that order, or all three of the following in this order:
   1. A valid non-negative integer\(^{669}\) followed by a U+0048 LATIN CAPITAL LETTER H character (H).
      The integer represents a duration of that number of hours.
   2. A valid non-negative integer\(^{669}\) followed by a U+004D LATIN CAPITAL LETTER M character (M).
      The integer represents a duration of that number of minutes.
   3. A valid non-negative integer\(^{669}\) followed by a U+0053 LATIN CAPITAL LETTER S character (S).
      The integer represents a duration of that number of seconds.

### 5.3.2.1 Conversion to iCalendar\(^{761}\)

Given a list of nodes \(\text{nodes}\) in a Document\(^{615}\), a user agent must run the following algorithm to extract any vEvent data represented by those nodes:

1. If none of the nodes in \(\text{nodes}\) are items with the type http://microformats.org/profile/hcalendar#vevent\(^{756}\), then there is no vEvent data. Abort the algorithm, returning nothing.
2. Let \(\text{output}\) be an empty string.
3. Add an iCalendar line\(^{762}\) with the type "BEGIN" and the value "VCALENDAR" to \(\text{output}\).
4. Add an iCalendar line\(^{762}\) with the type "PRODID" and the value equal to a user-agent-specific string representing the user agent to \(\text{output}\).
5. Add an iCalendar line\(^{762}\) with the type "VERSION" and the value "2.0" to \(\text{output}\).
6. For each node \(\text{node}\) in \(\text{nodes}\) that is an item with the type http://microformats.org/profile/hcalendar#vevent\(^{756}\), run the following steps:
   1. Add an iCalendar line\(^{762}\) with the type "BEGIN" and the value "VEVENT" to \(\text{output}\).
   2. Add an iCalendar line\(^{762}\) with the type "DTSTAMP" and a value consisting of an iCalendar DATE-TIME string representing the current date and time, with the annotation "VALUE=DATE-TIME", to \(\text{output}\). [RFC5545]\(^{1275}\)
   3. For each element \(\text{element}\) that is a property of the item \(\text{node}\) for each name \(\text{name}\) in element's property names\(^{739}\), run the appropriate set of substeps from the following list:
      - **If the property's value\(^{740}\) is an item**
        - Skip the property.
      - **If the property is dtend\(^{758}\)**
      - **If the property is dtstart\(^{759}\)**
      - **If the property is exdate\(^{759}\)**
      - **If the property is rdate\(^{760}\)**
      - **If the property is created\(^{760}\)**
      - **If the property is last-modified\(^{760}\)**
        - Let \(\text{value}\) be the result of stripping all U+002D HYPHEN-MINUS (-) and U+003A COLON (:) characters from the property's \(\text{value}\).
          - If the property's \(\text{value}\) is a valid date string\(^{735}\) then add an iCalendar line\(^{762}\) with the type \(\text{name}\) and the value \(\text{value}\) to \(\text{output}\), with the annotation "VALUE=DATE".
          - Otherwise, if the property's \(\text{value}\) is a valid global date and time string\(^{729}\) then add an iCalendar line\(^{762}\) with the type \(\text{name}\) and the value \(\text{value}\) to \(\text{output}\), with the annotation "VALUE=DATE-TIME".
        - Otherwise skip the property.
      - **Otherwise**
        - Add an iCalendar line\(^{762}\) with the type \(\text{name}\) and the property's \(\text{value}\) to \(\text{output}\).
   4. Add an iCalendar line\(^{762}\) with the type "END" and the value "VEVENT" to \(\text{output}\).
7. **Add an iCalendar line** with the type "END" and the value "VCALENDAR" to **output**.

When the above algorithm says that the user agent is to **add an iCalendar line** consisting of a type **type**, a value **value**, and optionally an annotation, to a string **output**, it must run the following steps:

1. Let **line** be an empty string.
2. Append **type**, converted to ASCII uppercase, to **line**.
3. If there is an annotation:
   1. Append a U+003B SEMICOLON character (;) to **line**.
   2. Append the annotation to **line**.
4. Append a U+003A COLON character (:) to **line**.
5. Prefix every U+005C REVERSE SOLIDUS character (\) in **value** with another U+005C REVERSE SOLIDUS character (\).
6. Prefix every U+002C COMMA character (,) in **value** with a U+005C REVERSE SOLIDUS character (\).
7. Prefix every U+003B SEMICOLON character (;) in **value** with a U+005C REVERSE SOLIDUS character (\).
8. Replace every U+000D CARRIAGE RETURN U+000A LINE FEED character pair (CRLF) in **value** with a U+005C REVERSE SOLIDUS character (\) followed by a U+006E LATIN SMALL LETTER N character (n).
9. Replace every remaining U+000D CARRIAGE RETURN (CR) or U+000A LINE FEED (LF) character in **value** with a U+005C REVERSE SOLIDUS character (\) followed by a U+006E LATIN SMALL LETTER N character (n).
10. Append **value** to **line**.
11. Let **maximum length** be 75.
12. While **line**'s code point length is greater than **maximum length**:
   1. Append the first **maximum length** code points of **line** to **output**.
   2. Remove the first **maximum length** code points from **line**.
   3. Append a U+000D CARRIAGE RETURN character (CR) to **output**.
   4. Append a U+000A LINE FEED character (LF) to **output**.
   5. Append a U+0020 SPACE character to **output**.
   6. Let **maximum length** be 74.
13. Append (what remains of) **line** to **output**.
14. Append a U+000D CARRIAGE RETURN character (CR) to **output**.
15. Append a U+000A LINE FEED character (LF) to **output**.

**Note**

*This algorithm can generate invalid iCalendar output, if the input does not conform to the rules described for the [http://microformats.org/profile/hcalendar#vevent][1] item type and defined property names.[2]*

### 5.3.2.2 Examples

This section is non-normative.

**Example**

Here is an example of a page that uses the vEvent vocabulary to mark up an event:

```html
<body itemscope itemtype="http://microformats.org/profile/hcalendar#vevent">
...
```

---

[1]: http://microformats.org/profile/hcalendar#vevent
[2]: http://microformats.org/profile/hcalendar#vevent
5.3.3 Licensing works

An item with the item type http://n.whatwg.org/work represents a work (e.g. an article, an image, a video, a song, etc). This type is primarily intended to allow authors to include licensing information for works.

The following are the type's defined property names.

work
Identifies the work being described.

The value must be an absolute URL.

Exactly one property with the name work must be present within each item with the type http://n.whatwg.org/work.

title
Gives the name of the work.

A single property with the name title may be present within each item with the type http://n.whatwg.org/work.

author
Gives the name or contact information of one of the authors or creators of the work.

The value must be either an item with the type http://microformats.org/profile/hcard, or text.

Any number of properties with the name author may be present within each item with the type http://n.whatwg.org/work.
license
Identifies one of the licenses under which the work is available.

The value must be an absolute URL.

Any number of properties with the name may be present within each item with the type http://n.whatwg.org/work.

5.3.3.1 Examples

This section is non-normative.

Example

This example shows an embedded image entitled My Pond, licensed under the Creative Commons Attribution-Share Alike 4.0 International License and the MIT license simultaneously.

```html
<figure itemscope itemtype=http://n.whatwg.org/work>
  <img itemprop="work" src="mypond.jpeg">
  <figcaption>
    <p><cite itemprop="title">My Pond</cite></p>
    <p><small>Licensed under the <a itemprop="license" href=https://creativecommons.org/licenses/by-sa/4.0/>Creative Commons Attribution-Share Alike 4.0 International License</a> and the <a itemprop="license" href=http://www.opensource.org/licenses/mit-license.php>MIT license</a>.</small>
  </figcaption>
</figure>
```

5.4 Converting HTML to other formats

5.4.1 JSON

Given a list of nodes nodes in a Document, a user agent must run the following algorithm to extract the microdata from those nodes into a JSON form:

1. Let result be an empty object.
2. Let items be an empty array.
3. For each node in nodes, check if the element is a top-level microdata item, and if it is then get the object for that element and add it to items.
4. Add an entry to result called "items" whose value is the array items.
5. Return the result of serializing result to JSON in the shortest possible way (meaning no whitespace between tokens, no unnecessary zero digits in numbers, and only using Unicode escapes in strings for characters that do not have a dedicated escape sequence), and with a lowercase "e" used, when appropriate, in the representation of any numbers. [JSON]

Note

This algorithm returns an object with a single property that is an array, instead of just returning an array, so that it is possible to extend the algorithm in the future if necessary.

When the user agent is to get the object for an item item, optionally with a list of elements memory, it must run the following substeps:

1. Let result be an empty object.
2. If no memory was passed to the algorithm, let memory be an empty list.
3. Add item to memory.
4. If the item has any item types\(^736\), add an entry to result called "type" whose value is an array listing the item types\(^736\) of item, in the order they were specified on the itemtype\(^736\) attribute.
5. If the item has a global identifier\(^737\), add an entry to result called "id" whose value is the global identifier\(^737\) of item.
6. Let properties be an empty object.
7. For each element element that has one or more property names\(^739\) and is one of the properties of the item\(^731\) item, in the order those elements are given by the algorithm that returns the properties of an item\(^731\), run the following substeps:
   1. Let value be the property value\(^740\) of element.
   2. If value is an item\(^736\), then: If value is in memory, then let value be the string "ERROR". Otherwise, get the object\(^764\) for value, passing a copy of memory, and then replace value with the object returned from those steps.
   3. For each name name in element's property names\(^739\), run the following substeps:
      1. If there is no entry named name in properties, then add an entry named name to properties whose value is an empty array.
      2. Append value to the entry named name in properties.
8. Add an entry to result called "properties" whose value is the object properties.
9. Return result.

Example
For example, take this markup:

```html
<!DOCTYPE HTML>
<html lang="en">
<title>My Blog</title>
<article itemscope itemtype="http://schema.org/BlogPosting">
  <header>
    <h1 itemprop="headline">Progress report</h1>
    <p><time itemprop="datePublished" datetime="2013-08-29">today</time></p>
    <link itemprop="url" href="#c1"/>
  </header>
  <p>All in all, he’s doing well with his swim lessons. The biggest thing was he had trouble putting his head in, but we got it down.</p>
</article>
<article itemscope itemtype="http://schema.org/UserComments" id="c1">
  <link itemprop="url" href="#c1"/>
  <footer>
    <p>Posted by: <span itemprop="creator" itemscope itemtype="http://schema.org/Person">
      <span itemprop="name">Greg</span>
    </span></p>
    <p><time datetime="2013-08-29">15 minutes ago</time></p>
  </footer>
  <p>Ha!</p>
</article>
<article itemscope itemtype="http://schema.org/UserComments" id="c2">
  <link itemprop="url" href="#c2"/>
  <footer>
    <p>Posted by: <span itemprop="creator" itemscope itemtype="http://schema.org/Person">
      <span itemprop="name">Charlotte</span>
    </span></p>
    <p><time datetime="2013-08-29">5 minutes ago</time></p>
  </footer>
  <p>When you say "we got it down"...</p>
```
It would be turned into the following JSON by the algorithm above (supposing that the page's URL was
https://blog.example.com/progress-report):

```json
{
  "items": [
    {
      "type": [ "http://schema.org/BlogPosting" ],
      "properties": {
        "headline": [ "Progress report" ],
        "datePublished": [ "2013-08-29" ],
        "url": [ "https://blog.example.com/progress-report?comments=0" ],
        "comment": [
          {
            "type": [ "http://schema.org/UserComments" ],
            "properties": {
              "url": [ "https://blog.example.com/progress-report#c1" ],
              "creator": [
                {
                  "type": [ "http://schema.org/Person" ],
                  "properties": {
                    "name": [ "Greg" ]
                  }
                }
              ],
              "commentTime": [ "2013-08-29" ]
            }
          },
          {
            "type": [ "http://schema.org/UserComments" ],
            "properties": {
              "url": [ "https://blog.example.com/progress-report#c2" ],
              "creator": [
                {
                  "type": [ "http://schema.org/Person" ],
                  "properties": {
                    "name": [ "Charlotte" ]
                  }
                }
              ],
              "commentTime": [ "2013-08-29" ]
            }
          }
        ]
    }
  ]
}```
6 User interaction

6.1 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 the Rendering section.

**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**

In the following skeletal example, the attribute is used to hide the web game's main screen until the user logs in:

```html
<h1>The Example Game</h1>
<section id="login">
<h2>Login</h2>
<form>
...<br/>
<!-- 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 id="game" hidden>
...<br/>
</section>
```

The `hidden` attribute must not be used to hide content that could legitimately be shown in another presentation. For example, it is incorrect to use `hidden` to hide panels in a tabbed dialog, because the tabbed interface is merely a kind of overflow presentation — one could equally well just show all the form controls in one big page with a scrollbar. It is similarly incorrect to use this attribute 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**

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 elements that they describe.
6.2 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 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
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 element is expressly inert if it is inert and its node document is not inert.

A Document document is blocked by a modal dialog subject if subject is the topmost dialog element in document's top layer. While document is so blocked, every node that is connected to document, with the exception of the subject element and its shadow, including 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.)

Note
The dialog element's showModal() method causes this mechanism to trigger, by adding the dialog element to its node document's top layer.

6.3 Tracking user activation

To prevent abuse of certain APIs that could be annoying to users (e.g., opening popups or vibrating phones), user agents allow these APIs only when the user is actively interacting with the web page or has interacted with the page at least once. This "active interaction" state is maintained through the mechanisms defined in this section.

6.3.1 Data model

For the purpose of tracking user activation, each Window has a last activation timestamp. This is a number indicating the last
time \( W \) got an activation notification\(^\text{p769}\). It corresponds to a `DOMHighResTimeStamp` value except for two cases: positive infinity indicates that \( W \) has never been activated, while negative infinity indicates that a user activation-gated API\(^\text{p770}\) has consumed\(^\text{p769}\) the last user activation of \( W \). The initial value is positive infinity.

A user agent also defines a transient activation duration, which is a constant number indicating how long a user activation is available for certain user activation-gated APIs\(^\text{p770}\) (e.g., for opening popups).

### Note

The transient activation duration\(^\text{p769}\) is expected be at most a few seconds, so that the user can possibly perceive the link between an interaction with the page and the page calling the activation-gated API.

These two values imply two boolean user activation states for \( W \):

**Sticky activation**

When the current high resolution time is greater than or equal to the last activation timestamp\(^\text{p768}\) in \( W \), \( W \) is said to have sticky activation\(^\text{p769}\).

This is \( W \)'s historical activation state, indicating whether the user has ever interacted in \( W \). It starts false, then changes to true (and never changes back to false) when \( W \) gets the very first activation notification\(^\text{p769}\).

**Transient activation**

When the current high resolution time is greater than or equal to the last activation timestamp\(^\text{p768}\) in \( W \) and less than the last activation timestamp\(^\text{p768}\) in \( W \) plus the transient activation duration\(^\text{p769}\), then \( W \) is said to have transient activation\(^\text{p769}\).

This is \( W \)'s current activation state, indicating whether the user has interacted in \( W \) recently. This starts with a false value, and remains true for a limited time after every activation notification\(^\text{p769}\) \( W \) gets.

The transient activation\(^\text{p769}\) state is considered expired if it becomes false because the transient activation duration\(^\text{p769}\) time has elapsed since the last user activation. Note that it can become false even before the expiry time through an activation consumption\(^\text{p769}\).

### 6.3.2 Processing model\(^\text{p769}\)

When a user interaction in a browsing context\(^\text{p811}\) \( B \) causes firing of an activation triggering input event\(^\text{p769}\) in \( B \)'s active document\(^\text{p813}\) \( D \), the user agent must perform the following activation notification steps before dispatching the event:

1. Let `browsingContexts` be a list consisting of:
   - \( B \),
   - all ancestor browsing contexts\(^\text{p816}\) of \( B \), and
   - all the descendant browsing contexts\(^\text{p817}\) of \( D \) that have active documents\(^\text{p813}\) from the same origin\(^\text{p841}\) as that of \( D \).
2. Let `windows` be the list of Window\(^\text{p827}\) objects constructed by taking the active window\(^\text{p813}\) of each item in `browsingContexts`.
3. For each window in `windows`, set window’s last activation timestamp\(^\text{p768}\) to the current high resolution time.

An activation triggering input event is any event whose `isTrusted` attribute is true and whose `type` is one of:

- `change`\(^\text{p1266}\)
- `click`
- `contextmenu`\(^\text{p1266}\)
- `dblclick`
- `mouseup`
- `pointerup`
- `reset`\(^\text{p1267}\)
- `submit`\(^\text{p1267}\)
- `touchend`

The event set is inconsistent across major browsers. See issue #3849.

Activation consuming APIs\(^\text{p770}\) defined in this and other specifications can consume user activation by performing the following...
steps, given a `Window` `W`:

1. If `W`'s `browsing context` is null, then return.
2. Let `top` be `W`'s `top-level browsing context`.
3. Let `browsingContexts` be the list of the descendant browsing contexts of `top`'s `active document`.
4. Append `top` to `browsingContexts`.
5. Let `windows` be the list of `Window` objects constructed by taking the `active window` of each item in `browsingContexts`.
6. For each `window` in `windows`, if `window`'s `last activation timestamp` is not positive infinity, then set `window`'s `last activation timestamp` to negative infinity.

The spec is not clear about how to traverse a tree of documents. See issue #5020.

**Note**

Note the asymmetry in the sets of `browsing contexts in the page that are affected by an activation notification` vs an `activation consumption`, an activation consumption changes (to false) the `transient activation` states for all browsing contexts in the page, but an activation notification changes (to true) the states for a subset of those browsing contexts. The exhaustive nature of consumption here is deliberate: it prevents malicious sites from making multiple calls to an `activation consuming API` from a single user activation (possibly by exploiting a deep hierarchy of `iframe`).

### 6.3.3 APIs gated by user activation

APIs that are dependent on user activation are classified into three different levels. The levels are as follows, sorted by their "strength of dependence" on user activation (from weakest to strongest):

**Sticky activation-gated APIs**

These APIs require the `sticky activation` state to be true, so they are blocked until the very first user activation.

**Transient activation-gated APIs**

These APIs require the `transient activation` state to be true, but they don't `consume` it, so multiple calls are allowed per user activation until the transient state `expires`.

**Transient activation-consuming APIs**

These APIs require the `transient activation` state to be true, and they `consume user activation` in each call to prevent multiple calls per user activation.

### 6.4 Activation behavior of elements

Certain elements in HTML have an `activation behavior`, which means that the user can activate them. This is always caused by a `click` event.

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 fire a `click event` at the element.

**For web developers (non-normative)**

```javascript
element .click()
```

Acts as if the element was clicked.

Each element has an associated `click in progress flag`, which is initially unset.

The `click()` method must run the following steps:
1. If this element is a form control that is disabled, then return.

2. If this element's click in progress flag is set, then return.

3. Set this element's click in progress flag.

4. Fire a synthetic pointer event named click at this element, with the not trusted flag set.

5. Unset this element's click in progress flag.

6.5 Focus

6.5.1 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, dialog boxes) 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

Consider an HTML application running in a browser tab running in a graphical environment. Suppose this application had a page with some text controls and links, and was currently showing a modal dialog, which itself had a text control 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 controls, as well as the dialog. The dialog itself would have as its children the text control and the button.

If the widget with focus in this example was the text control 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 control.
Keyboard events are always targeted at this focused element.

### 6.5.2 Data model

A top-level browsing context has system focus when it can receive keyboard input channeled from the operating system.

**Note**

System focus is lost when a browser window loses focus, but might also be lost to other system widgets in the browser window such as a URL bar.

The term focusable area is used to refer to regions of the interface that can further become the target of such 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.)

<table>
<thead>
<tr>
<th>Focusable area</th>
<th>DOM anchor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements that meet all the following criteria:</td>
<td>The element itself.</td>
</tr>
<tr>
<td>- the element's tabindex value is non-null, or the element is determined by the user agent to be focusable;</td>
<td></td>
</tr>
<tr>
<td>- the element is either not a shadow host, or has a shadow root whose delegates focus is false;</td>
<td></td>
</tr>
<tr>
<td>- the element is not actually disabled;</td>
<td></td>
</tr>
<tr>
<td>- the element is not expressly inert;</td>
<td></td>
</tr>
<tr>
<td>- the element is either being rendered or being used as relevant canvas fallback content.</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>iframe, \textless input type=text\textgreater , sometimes \textless a href=&quot;\textgreater (depending on platform conventions).</td>
</tr>
<tr>
<td>The shapes of area elements in an image map associated with an img element that is being rendered and is not expressly inert.</td>
<td>The img element.</td>
</tr>
<tr>
<td>Example</td>
<td>In the following example, the area element creates two shapes, one on each image. The DOM anchor of the first shape is the first img element, and the DOM anchor of the second shape is the second img element.</td>
</tr>
<tr>
<td>The user-agent provided subwidgets of elements that are being rendered and are not actually disabled, or expressly inert.</td>
<td>The element for which the focusable area is a subwidget.</td>
</tr>
<tr>
<td>Example</td>
<td>The controls in the user interface for a video element, the up and down buttons in a spin-control version of, the part of a details element's rendering that enables the element to be opened or closed using keyboard input.</td>
</tr>
<tr>
<td>The scrollable regions of elements that are being rendered and are not expressly inert.</td>
<td>The element for which the box that the scrollable region scrolls was created.</td>
</tr>
</tbody>
</table>
### Focusable area

**Examples**

- **Example**
  - The viewport of a Document that has a non-null browsing context and is not inert.
  - The Document, for which the viewport was created.

- **Example**
  - The contents of an iframe.

- **Example**
  - Any other element or part of an element determined by the user agent to be a focusable area, especially to aid with accessibility or to better match platform conventions.
  - The element.

- **Example**
  - A user agent could make all list item bullets sequentially focusable, so that a user can more easily navigate lists.

- **Example**
  - Similarly, a user agent could make all elements with title attributes sequentially 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 its 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.

---

One focusable area in each Document is designated the **focused area of the document**. Which control is so designated changes over time, based on algorithms in this specification.

The **currently focused area** of a top-level browsing context `topLevelBC` at any particular time is the focusable area, or-null returned by this algorithm:

1. If `topLevelBC` does not have system focus, then return null.
2. Let `candidate` be `topLevelBC`'s active document.
3. While `candidate`'s focused area is a browsing context container with a non-null nested browsing context, set `candidate` to the active document of that browsing context's nested browsing context.
4. If `candidate`'s focused area is non-null, set `candidate` to `candidate`'s focused area.
5. Return `candidate`.

The **current focus chain** of a top-level browsing context `topLevelBC` at any particular time is the focus chain of the currently focused area of `topLevelBC`, if `topLevelBC` is non-null, or an empty list otherwise.

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 subject is the ordered list constructed as follows:

1. Let `output` be an empty list.
2. Let `currentObject` be `subject`.
3. While true:
   1. Append `currentObject` to `output`.

---

Example of the CSS `overflow` property’s `scroll` value typically creates a scrollable region.

Example of the contents of an iframe.

Example of a user agent could make all list item bullets sequentially focusable, so that a user can more easily navigate lists.

Example of a user agent could make all elements with title attributes sequentially focusable, so that their advisory information can be accessed.

Example of A browsing context container is a focusable area, but key events routed to a browsing context container get immediately routed to its 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.

---

773
2. If `currentObject` is an area element's shape, then append that area element to output. Otherwise, if `currentObject`'s DOM anchor is an element that is not `currentObject` itself, then append `currentObject`'s DOM anchor to output.

3. If `currentObject` is a focusable area, then set `currentObject` to `currentObject`'s DOM anchor's node document. Otherwise, if `currentObject` is a Document whose browsing context is a child browsing context, then set `currentObject` to `currentObject`'s browsing context's container.

   Otherwise, break.

4. Return output.

All elements that are focusable areas are said to be focusable.

There are two special types of focusability for focusable areas:

- A focusable area is said to be sequentially focusable if it is included in its Document's sequential focus navigation order and the user agent determines that it is sequentially focusable.
- A focusable area is said to be click focusable if the user agent determines that it is click focusable. User agents should consider focusable areas with non-null `tabindex` values to be click focusable.

A node is a focus navigation scope owner if it is a document, a shadow host or a slot.

Each focus navigation scope owner has a focus navigation scope, which is a list of elements. Its contents are determined as follows:

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.

All elements that are focusable areas are said to be focusable.
Every element `element` has an **associated focus navigation owner**, which is either null or a `focus navigation scope owner`. It is determined by the following algorithm:

1. If `element`'s parent is null, then return null.
2. If `element`'s parent is a `shadow host`, then return `element`'s **assigned slot**.
3. If `element`'s parent is a `shadow root`, then return the parent's **host**.
4. If `element`'s parent is the `document element`, then return the parent's **node document**.
5. Return `element`'s parent's **associated focus navigation owner**.

Then, the contents of a given `focus navigation scope owner`'s **focus navigation scope** are all elements whose **associated focus navigation owner** is owner.

**Note**

The order of elements within a `focus navigation scope` does not impact any of the algorithms in this specification. Ordering only becomes important for the **tabindex-ordered focus navigation scope** and **flattened tabindex-ordered focus navigation scope** concepts defined below.

A **tabindex-ordered focus navigation scope** is a list of `focusable areas` and `focus navigation scope owners`. Every `focus navigation scope owner` owner has a **tabindex-ordered focus navigation scope**, whose contents are determined as follows:

- It contains all elements in owner's `focus navigation scope` that are themselves `focus navigation scope owners`, except the elements whose **tabindex value** is a negative integer.
- It contains all of the `focusable areas`, whose **DOM anchor** is an element in owner's `focus navigation scope`, except the `focusable areas` whose **tabindex value** is a negative integer.

The order within a **tabindex-ordered focus navigation scope** is determined by each element's **tabindex value**, as described in the section below.

**Note**

The rules there do not give a precise ordering, as they are composed mostly of "should" statements and relative orderings.

A **flattened tabindex-ordered focus navigation scope** is a list of `focusable areas`. Every `focus navigation scope owner` owner owns a distinct **flattened tabindex-ordered focus navigation scope**, whose contents are determined by the following algorithm:

1. Let `result` be a **clone** of owner's **tabindex-ordered focus navigation scope**.
2. For each item of `result`:
   1. If `item` is not a `focus navigation scope owner`, then **continue**.
   2. If `item` is not a `focusable area`, then replace `item` with all of the items in `item`'s **flattened tabindex-ordered focus navigation scope**.
   3. Otherwise, insert the contents of `item`'s **flattened tabindex-ordered focus navigation scope** after `item`.

6.5.3 The **tabindex** attribute

The **tabindex** content attribute allows authors to make an element and regions that have the element as its **DOM anchor** be `focusable areas`, allow or prevent them from being **sequentially focusable**, and determine their relative ordering for **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 **sequentially focusable** `focusable areas`.

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 not
Developers should use caution when using values other than 0 or −1 for their `tabindex` attributes as this is complicated to do correctly.

**Note**

The following provides a non-normative summary of the behaviors of the possible `tabindex` attribute values. The below processing model gives the more precise rules.

**omitted (or non-integer values)**

The user agent will decide whether the element is focusable, and if it is, whether it is sequentially focusable or click focusable (or both).

**−1 (or other negative integer values)**

Causes the element to be focusable, and indicates that the author would prefer the element to be click focusable but not sequentially focusable. The user agent might ignore this preference for click and sequential focusability, e.g., for specific element types according to platform conventions, or for keyboard-only users.

**0**

Causes the element to be focusable, and indicates that the author would prefer the element to be both click focusable and sequentially focusable. The user agent might ignore this preference for click and sequential focusability.

**positive integer values**

Behaves the same as 0, but in addition creates a relative ordering within a `tabindex-ordered focus navigation scope`, so that elements with higher `tabindex` attribute value come later.

Note that the `tabindex` attribute cannot be used to make an element non-focusable. The only way a page author can do that is by disabling the element, or making it inert.

The `tabindex` of an element is the value of its `tabindex` attribute, parsed using the rules for parsing integers. If parsing fails or the attribute is not specified, then the `tabindex` is null.

The `tabindex` of a focusable area is the `tabindex` of its DOM anchor.

The `tabindex` of an element must be interpreted as follows:

**If the value is null**

The user agent should follow platform conventions to determine if the element should be considered as a focusable area and if so, whether the element and any focusable areas that have the element as their DOM anchor are sequentially focusable, and if so, what their relative position in their `tabindex-ordered focus navigation scope` is to be. If the element is a focus navigation scope owner, it must be included in its `tabindex-ordered focus navigation scope`, even if it is not a focusable area.

The relative ordering within a `tabindex-ordered focus navigation scope` for elements and focusable areas that belong to the same focus navigation scope and whose `tabindex value` is null should be in shadow-including tree order.

Modulo platform conventions, it is suggested that the following elements should be considered as focusable areas and be sequentially focusable:

- elements that have an `href` attribute
- `button` elements
- elements whose `type` attribute are not in the `Hidden` state
- `select` elements
- `textarea` elements
- elements that are the first `summary` element child of a `details` element
- Elements with a `draggable` attribute set, if that would enable the user agent to allow the user to begin drag operations for those elements without the use of a pointing device
• Editing hosts
• Browsing context containers

If the value is a negative integer
The user agent must consider the element as a focusable area, but should omit the element from any tabindex-ordered focus navigation scope.

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 control 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 allow the element to be considered as a focusable area and should allow the element and any focusable areas that have the element as their DOM anchor to be sequentially focusable.

The relative ordering within a tabindex-ordered focus navigation scope for elements and focusable areas that belong to the same focus navigation scope, and whose tabindex value is zero should be in shadow-including tree order.

If the value is greater than zero
The user agent must allow the element to be considered as a focusable area and should allow the element and any focusable areas that have the element as their DOM anchor to be sequentially focusable, and should place the element — referenced as candidate below — and the aforementioned focusable areas in the tabindex-ordered focus navigation scope where the element is a part of so that, relative to other elements and focusable areas that belong to the same focus navigation scope, 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 located earlier than candidate in shadow-including tree order,
• 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 located later than candidate in shadow-including tree order, 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.

The tabindex IDL attribute must reflect the value of the tabindex content attribute. The default value is 0 if the element is an a, area, button, frame, iframe, input, object, select, textarea, or SVG a element, or is a summary element that is a summary for its parent details. The default value is −1 otherwise.

Note
The varying default value based on element type is a historical artifact.

6.5.4 Processing model

To get the focusable area for a focus target that is either an element that is not a focusable area, or is a browsing context, given an optional string focus trigger, run the first matching set of steps from the following list:

If focus target is an area element with one or more shapes that are focusable areas

Return the shape corresponding to the first img element in tree order that uses the image map to which the area element...
If focus target is an element with one or more scrollable regions that are focusable areas\(^{772}\)
Return the element's first scrollable region, according to a pre-order, depth-first traversal of the flat tree. [CSSSCOPING]\(^{1272}\)

If focus target is the document element of its Document\(^{p113}\)
Return the Document\(^{p113}\)'s viewport.

If focus target is a browsing context\(^{813}\)
Return the browsing context\(^{813}\)'s active document\(^{813}\).

If focus target is a browsing context container\(^{816}\) with a non-null nested browsing context\(^{816}\)
Return the browsing context container\(^{816}\)'s nested browsing context\(^{816}\)'s active document\(^{813}\).

If focus target is a shadow host whose shadow root's delegates focus is true
1. If focus target is a shadow-including inclusive ancestor of the currently focused area of a top-level browsing context\(^{p773}\)'s DOM anchor\(^{p772}\), then return null.
2. Otherwise:
   1. If focus trigger is "click", then let possible focus delegates be the list of all click focusable\(^{p774}\) focusable areas\(^{p772}\) whose DOM anchor\(^{p772}\) is a descendant of focus target in the flat tree.
   2. Otherwise, let possible focus delegates be the list of all focusable areas\(^{p772}\) whose DOM anchor\(^{p772}\) is a descendant of focus target in the flat tree.
   3. Return the first focusable area\(^{p772}\) in tree order of their DOM anchors\(^{p772}\) in possible focus delegates, or null if possible focus delegates is empty.

Otherwise
Return null.

The focusing steps for an object new focus target that is either a focusable area\(^{p772}\), or an element that is not a focusable area\(^{p772}\), or a browsing context\(^{813}\), are as follows. They can optionally be run with a fallback target and a string focus trigger.

1. If new focus target is not a focusable area\(^{p772}\), then set new focus target to the result of getting the focusable area\(^{p777}\) for new focus target, given focus trigger if it was passed.
2. If new focus target is null, then:
   1. If no fallback target was specified, then return.
   2. Otherwise, set new focus target to the fallback target.
3. If new focus target is a browsing context container\(^{816}\) with non-null nested browsing context\(^{816}\), then set new focus target to the nested browsing context\(^{816}\)'s active document\(^{813}\).
4. If new focus target is a focusable area\(^{p772}\) and its DOM anchor\(^{p772}\) is inert\(^{p768}\), then return.
5. If new focus target is the currently focused area of a top-level browsing context\(^{p773}\), then return.
6. Let old chain be the current focus chain of the top-level browsing context\(^{p773}\) in which new focus target finds itself.
7. Let new chain be the focus chain\(^{p773}\) of new focus target.
8. Run the focus update steps\(^{779}\) with old chain, new chain, and new focus target respectively.

User agents must immediately\(^{542}\) run the focusing steps\(^{778}\) for a focusable area\(^{p772}\) or browsing context\(^{813}\) 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 return.
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 current focus chain of the top-level browsing context in which old focus target finds itself.
4. If old focus target is not one of the entries in old chain, then return.
5. If old focus target is not a focusable area, then return.
6. Let topDocument be old chain's last entry.
7. If topDocument's browsing context has system focus, then run the focusing steps for topDocument's viewport.
   Otherwise, apply any relevant platform-specific conventions for removing system focus from topDocument's browsing context, and run the focus update steps with old chain, an empty list, and null respectively.

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.

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.

### Focus fixup rule

When the designated focusable area of the document is removed from that Document in some way (e.g. it stops being a focusable area, it is removed from the DOM, it becomes expressly inert, etc.), designate the Document's viewport to be the new focusable area of the document.

### Example

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.

### Example

In a Document, whose focusable area is a button element, removing, disabling, or hiding that button would cause the page's new focusable area to be the viewport of the Document. This would, in turn, be reflected through the activeElement API as the body element.

The focus update steps, given an old chain, a new chain, and a new focus target respectively, are as follows:

1. 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.
2. For each 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 (such that it is different to what it was when the control was first focused), then fire an event named change at the element, with the bubbles attribute initialized to true.
   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 relevant global 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.

3. Apply any relevant platform-specific conventions for focusing new focus target. (For example, some platforms select the contents of a text control when that control is focused.)

4. 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 document.
   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 relevant global 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.

To fire a focus event named e at an element t with a given related target r, fire an event named e at t, using FocusEvent, with the relatedTarget attribute initialized to r, the view attribute initialized to t's node document's relevant global object, and the composed flag set.

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. Assert: target area is not null, since key events are only routed to top-level browsing contexts that have system focus.
3. If target area is a focusable area, let target node be target area's DOM anchor. Otherwise, target area is a dialog; let target node be target area.
4. 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 object that has a non-null document element, then let target node be that document element.
5. If target node is not inert, then:

   Note
   It is possible for the currently focused area of a top-level browsing context to be inert, for example if a modal dialog is shown, and then that dialog element is made inert. It is likely to be the result of a logic error in the application, though.

   1. Let canHandle be the result of dispatching the key event at target node.
   2. If canHandle is true, then let target area handle the key event. This might include firing a click event at target node.

The has focus steps, given a Document object target, are as follows:
1. If target's browsing context's top-level browsing context does not have system focus, then return false.
2. Let candidate be target's top-level browsing context's active document.
3. While true:
   1. If candidate is target, then return true.
   2. If the focused area of candidate is a browsing context container with a non-null nested browsing context, then set candidate to the active document of that browsing context container's nested browsing context.
   3. Otherwise, return false.

6.5.5 Sequential focus navigation

Each Document has a sequential focus navigation order, which orders some or all of the focusable areas in the Document relative to each other. Its contents and ordering are given by the flattened tabindex-ordered focus navigation scope of the Document.

Note
Per the rules defining the flattened tabindex-ordered focus navigation scope, the ordering is not necessarily related to the tree order of the Document.

If a focusable area is omitted from the sequential focus navigation order of its Document, 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
For example, the user agent could set it to the position of the user's click if the user clicks on the document contents.

Note
User agents are required to set the sequential focus navigation starting point to the target element when navigating to a fragment.

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 Document's sequential focus navigation order.

   Otherwise, starting point is not in its Document'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 return.
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 return.

Example
For example, if direction is backward, then the last sequentially focusable control before the browser’s rendering area would be the control to focus.

If the user agent has no sequentially 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.

9. Otherwise, starting point is a focusable area in a child browsing context. Set starting point to that child browsing context’s 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.

1. 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.

<table>
<thead>
<tr>
<th>starting point is a browsing context</th>
<th>direction is forward</th>
<th>direction is backward</th>
</tr>
</thead>
<tbody>
<tr>
<td>selection mechanism is DOM</td>
<td>Let candidate be the first suitable sequentially focusable area in starting point’s active document, if any; or else null</td>
<td>Let candidate be the last suitable sequentially focusable area in starting point’s active document, if any; or else null</td>
</tr>
<tr>
<td>selection mechanism is sequential</td>
<td>Let candidate be the first suitable sequentially focusable area in the home document following starting point, if any; or else null</td>
<td>Let candidate be the last suitable sequentially focusable area in the home document preceding starting point, if any; or else null</td>
</tr>
</tbody>
</table>

A suitable sequentially focusable area is a focusable area whose DOM anchor is not inert and is sequentially focusable.

The home document is the Document 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 document’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).

2. If candidate is a browsing context container with a non-null nested browsing context, 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.

3. Return candidate.

6.5.6 Focus management APIs

```idl
dictionary FocusOptions {
  boolean preventScroll = false;
};
```

For web developers (non-normative)
The `activeElement` attribute's getter must run these steps:

1. Let `candidate` be the DOM anchor of the focused area of this `DocumentOrShadowRoot`'s node document.
2. Set `candidate` to the result of retargeting `candidate` against this `DocumentOrShadowRoot`.
3. If `candidate`'s root is not this `DocumentOrShadowRoot`, then return null.
4. If `candidate` is not a `Document` object, then return `candidate`.
5. If `candidate` has a body element, then return that body element.
6. If `candidate`'s document element is non-null, then return that document element.
7. Return null.

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, when invoked, must run these steps:

1. Let `current` be this `Window` object's browsing context.
2. If `current` is null, then return.
3. Run the focusing steps with current.

4. If current 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, when invoked, provides a hint to the user agent that the script believes the user probably is not currently interested in the contents of this Window's browsing context, if non-null, 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(options) method on elements, when invoked, must run the following steps:

1. If the element is marked as locked for focus, then return.
2. Mark the element as locked for focus.
3. Run the focusing steps for the element.
4. If the value of the preventScroll dictionary member of options is false, then scroll the element into view with scroll behavior "auto", block flow direction position set to an implementation-defined value, and inline base direction position set to an implementation-defined value.
5. 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**

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.

### 6.5.7 The autofocus attribute

The autofocus content attribute allows the author to indicate that an element is to be focused as soon as the page is loaded or as soon as the dialog within which it finds itself is shown, allowing the user to just start typing without having to manually focus the main element.

The autofocus attribute is a boolean attribute.

An element's nearest ancestor autofocus scoping root element is the element itself if the element is a dialog element, or else is the element's nearest ancestor dialog element, if any, or else is the element's last inclusive ancestor element.

There must not be two elements with the same nearest ancestor autofocus scoping root element that both have the autofocus attribute specified.

Each Document has an autofocus candidates list, initially empty.

Each Document has an autofocus processed flag boolean, initially false.

When an element with the autofocus attribute specified is inserted into a document, run the following steps:

1. If the user has indicated (for example, by starting to type in a form control) that they do not wish focus to be changed, then optionally return.
2. Let target be the element's node document.
3. If target's browsing context is null, then return.
4. If target's active sandboxing flag set has the sandboxed automatic features browsing context flag, then return.

5. Let topDocument be the active document of target's browsing context's top-level browsing context.

6. If target's origin is not the same as the origin of topDocument, then return.

7. If topDocument's autofocus processed flag is false, then remove the element from topDocument's autofocus candidates, and append the element to topDocument's autofocus candidates.

To flush autofocus candidates for a document topDocument, run these steps:

1. If topDocument's autofocus processed flag is true, then return.

2. Let candidates be topDocument's autofocus candidates.

3. If candidates is empty, then return.

4. If topDocument's focused area is not topDocument itself, or topDocument's URL's fragment is not empty, then:
   1. Empty candidates.
   2. Set topDocument's autofocus processed flag to true.
   3. Return.

5. While candidates is not empty:
   1. Let element be candidates[0].
   2. Let doc be element's node document.
   3. If doc is not fully active, then remove element from candidates, and continue.
   4. If doc's browsing context is top-level browsing context is not same as topDocument's browsing context, then remove element from candidates, and continue.
   5. If doc's script-blocking style sheet counter is greater than 0, then return.

6. Remove element from candidates.

7. Let inclusiveAncestorDocuments be a list consisting of doc, plus the active documents of each of doc's ancestor browsing contexts.

8. If URL's fragment of any Document in inclusiveAncestorDocuments is not empty, then continue.

9. Let target be element.

10. If target is not a focusable area, then set target to the result of getting the focusable area for target.

Note: Autofocus candidates can contain elements which are not focusable areas. In addition to the special cases handled in the get the focusable area algorithm, this can happen because a non-focusable area element with an autofocus attribute was inserted into a document and it never became focusable, or because the element was focusable but its status changed while it was stored in autofocus candidates.
3. Run the focusing steps for target.

**Note**

This handles the automatic focusing during document load. The `show()` and `showModal()` methods of `dialog` elements also processes the autofocus attribute.

**Note**

Focusing the element does not imply that the user agent has to focus the browser window if it has lost focus.

The autofocus IDL attribute must reflect the content attribute of the same name.

**Example**

In the following snippet, the text control would be focused when the document was loaded.

```html
<input maxlength="256" name="q" value="" autofocus>
<input type="submit" value="Search">
```

**Example**

The autofocus attribute applies to all elements, not just to form controls. This allows examples such as the following:

```html
<div contenteditable autofocus> Edit <strong>me!</strong></div>
```

### 6.6 Assigning keyboard shortcuts

**6.6.1 Introduction**

This section is non-normative.

Each element that can be activated or focused can be assigned a single key combination to activate it, using the accesskey attribute.

The exact shortcut is determined by the user agent, based on information about the user's keyboard, what keyboard shortcuts already exist on the platform, and what other shortcuts have been specified on the page, using the information provided in the accesskey attribute as a guide.

In order to ensure that a relevant keyboard shortcut is available on a wide variety of input devices, the author can provide a number of alternatives in the accesskey attribute.

Each alternative 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. The accessKeyLabel IDL attribute returns a string representing the actual key combination assigned by the user agent.

**Example**

In this example, an author has provided a button that can be invoked using a shortcut key. To support full keyboards, the author has provided "C" as a possible key. To support devices equipped only with numeric keypads, the author has provided "1" as another possibly key.

```html
<input type=button value=Collect onclick="collect()"
accesskey="C 1" id=c>
```

**Example**

To tell the user what the shortcut key is, the author has this script here opted to explicitly add the key combination to the button's label:
function addShortcutKeyLabel(button) {
  if (button.accessKeyLabel != '')
    button.value += ' (' + button.accessKeyLabel + ')';
} addShortcutKeyLabel(document.getElementById('c'));

Browsers on different platforms will show different labels, even for the same key combination, based on the convention prevalent on that platform. For example, if the key combination is the Control key, the Shift key, and the letter C, a Windows browser might display "Ctrl+Shift+C", whereas a Mac browser might display "^C", while an Emacs browser might just display "C-C". Similarly, if the key combination is the Alt key and the Escape key, Windows might use "Alt+Esc", Mac might use "⌥Esc", and an Emacs browser might use "M-ESCAPE" or "ESC ESC".

In general, therefore, it is unwise to attempt to parse the value returned from the accessKeyLabel IDL attribute.

### 6.6.2 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 an ordered set of unique space-separated tokens, none of which are identical to another token and each of which must be exactly one code point in length.

**Example**

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:

```html
<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>
```

**Example**

In the following example, the search field is given two possible access keys, "s" and "0" (in that order). A user agent on a device with a full keyboard might pick Ctrl+Alt+s as the shortcut key, while a user agent on a small device with just a numeric keypad might pick just the plain unadorned key 0:

```html
<form action="/search">
  <label>Search: <input type="search" name="q" accesskey="s 0"></label>
  <input type="submit">
</form>
```

**Example**

In the following example, a button has possible access keys described. A script then tries to update the button’s label to advertise the key combination the user agent selected.

```html
<input type="submit accesskey="N @ 1" value="Compose"> ...
<script>
```
An element's assigned access key is a key combination derived from the element's accesskey content attribute. 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. Otherwise, split the attribute's value on ASCII whitespace, and let keys be the resulting tokens.
3. For each value in keys in turn, in the order the tokens appeared in the attribute's value, run the following substeps:
   1. If the value is not a string exactly one code point in length, then skip the remainder of these steps for this value.
   2. If the value does not correspond to a key on the system's keyboard, then skip the remainder of these steps for this value.
   3. If the user agent can find a mix of zero or more modifier keys that, combined with the key that corresponds to the value given in the attribute, can be used as the access key, then the user agent may assign that combination of keys as the element's assigned access key and return.
4. **Fallback**: Optionally, the user agent may assign a key combination of its choosing as the element's assigned access key and then return.
5. 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 agent processes 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 a non-null 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.

The accessKeyLabel IDL attribute must return a string that represents the element's assigned access key, if any. If the element does not have one, then the IDL attribute must return the empty string.
IDL  
interface mixin ElementContentEditable {
    [CEReactions] attribute DOMString contentEditable;
    [CEReactions] attribute DOMString enterKeyHint;
    readonly attribute boolean isContentEditable;
    [CEReactions] attribute DOMString inputMode;
};

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.

Example

For example, consider a page that has a `form` and a `textarea` to publish a new article, where the user is expected to write the article using HTML:

```html
<form method="POST">
  <fieldset>
    <legend>New article</legend>
    <textarea name="article">Hello world.</textarea>
  </fieldset>
  <p><button>Publish</button></p>
</form>
```

When scripting is enabled, the `textarea` element could be replaced with a rich text control instead, using the `contenteditable` attribute:

```html
<form method="POST">
  <fieldset>
    <legend>New article</legend>
    <textarea id="textarea" name="article">Hello world.</textarea>
    <div id="div" style="white-space: pre-wrap" hidden><p>Hello world.</p></div>
    <script>
      let textarea = document.getElementById("textarea");
      let div = document.getElementById("div");
      textarea.hidden = true;
      div.hidden = false;
      div.contentEditable = "true";
      div.oninput = (e) => {
        textarea.value = div.innerHTML;
      };
    </script>
  </fieldset>
  <p><button>Publish</button></p>
</form>
```

Features to enable, e.g., inserting links, can be implemented using the `document.execCommand()` API, or using `Selection` APIs and other DOM APIs.

Example

The `contenteditable` attribute can also be used to great effect:

```html
<!doctype html>
<html lang="en">
```

---

[Editorial Note: 6.7 Editing](#)
The contentEditable IDL attribute, on getting, must return the string "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.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 isContentEditable IDL attribute, on getting, must return true if the element is either an editing host or editable, and false otherwise.

6.7.2 Making entire documents editable: the designMode getter and setter

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.

Document objects have an associated design mode enabled, which is a boolean. It is initially false.

The designMode getter steps are to return "on" if this's design mode enabled is true; otherwise "off".

The designMode setter steps are:

1. Let value be the given value, converted to ASCII lowercase.
2. If value is "on" and this's design mode enabled is false, then:
   1. Set this's design mode enabled to true.
   2. Reset this's active range's start and end boundary points to be at the start of this.
   3. Run the focusing steps for this's document element, if non-null.
3. If value is "off", then set this's design mode enabled to false.

6.7.3 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**

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.

### 6.7.4 Editing APIs

An **editing host** is either an **HTML element** with its `contenteditable` attribute in the **true** state, or a **child** **HTML element** of a **Document** whose **design mode enabled** is **true**.

The definition of the terms **active range**, **editing host of**, 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 defined in **execCommand**.

### 6.7.5 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.
For web developers (non-normative)

**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.

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 **Email** 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 UA 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. UAs 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.
Some methods of entering text, for example virtual keyboards on mobile devices, and also voice input, often assist users by automatically capitalizing the first letter of sentences (when composing text in a language with this convention). A virtual keyboard that implements autocapitalization might automatically switch to showing uppercase letters (but allow the user to toggle it back to lowercase) when a letter that should be autocapitalized is about to be typed. Other types of input, for example voice input, may perform autocapitalization in a way that does not give users an option to intervene first. The `autocapitalize` attribute allows authors to control such behavior.

The `autocapitalize` attribute, as typically implemented, does not affect behavior when typing on a physical keyboard. (For this reason, as well as the ability for users to override the autocapitalization behavior in some cases or edit the text after initial input, the attribute must not be relied on for any sort of input validation.)

The `autocapitalize` attribute can be used on an `editing host` to control autocapitalization behavior for the hosted editable region, on an `input` or `textarea` element to control the behavior for inputting text into that element, or on a `form` element to control the default behavior for all `autocapitalize-inheriting elements` associated with the `form` element.

The `autocapitalize` attribute never causes autocapitalization to be enabled for `input` elements whose `type` attribute is in one of the `URL`, `Email`, or `Password` states. (This behavior is included in the `used autocapitalization hint` algorithm below.)

The autocapitalization processing model is based on selecting among five `autocapitalization hints`, defined as follows:

- **default**
  The user agent and input method should use make their own determination of whether or not to enable autocapitalization.

- **none**
  No autocapitalization should be applied (all letters should default to lowercase).

- **sentences**
  The first letter of each sentence should default to a capital letter; all other letters should default to lowercase.

- **words**
  The first letter of each word should default to a capital letter; all other letters should default to lowercase.

- **characters**
  All letters should default to uppercase.

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.

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.

```html
<div contenteditable="true">
  <span spellcheck="false" id="a">Hell</span>o!
</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).

```html
<p spellcheck="true">
  <label>Name: <input spellcheck="false" id="b"></label>
</p>
```
hint\textsuperscript{794}, which informs the behavior of the user agent. The keywords for this attribute and their state mappings are as follows:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>none\textsuperscript{793}</td>
</tr>
<tr>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>on</td>
<td>sentences\textsuperscript{793}</td>
</tr>
<tr>
<td>sentences</td>
<td>words\textsuperscript{791}</td>
</tr>
<tr>
<td>words</td>
<td>characters\textsuperscript{793}</td>
</tr>
</tbody>
</table>

The invalid value default\textsuperscript{968} is the sentences\textsuperscript{793} state. The missing value default\textsuperscript{968} is the default\textsuperscript{793} state.

\textbf{For web developers (non-normative)}

\texttt{element.autocapitalize\textsuperscript{794} [ = value ]}

Returns the current autocapitalization state for the element, or an empty string if it hasn't been set. Note that for \texttt{input}\textsuperscript{493} and \texttt{textarea}\textsuperscript{548} elements that inherit their state from a \texttt{form}\textsuperscript{486} element, this will return the autocapitalization state of the \texttt{form}\textsuperscript{486} element, but for an element in an editable region, this will not return the autocapitalization state of the editing host (unless this element is, in fact, the editing host\textsuperscript{791}). Can be set, to set the autocapitalize\textsuperscript{794} content attribute (and thereby change the autocapitalization behavior for the element).

To compute the own autocapitalization hint of an element element, run the following steps:

1. If the autocapitalize\textsuperscript{794} content attribute is present on element, and its value is not the empty string, return the state of the attribute.
2. If element is an autocapitalize-inheriting element\textsuperscript{486} and has a non-null form owner\textsuperscript{566}, return the own autocapitalization hint\textsuperscript{794} of element's form owner\textsuperscript{566}.
3. Return default\textsuperscript{793}.

The autocapitalize getter steps are to:

1. Let state be the own autocapitalization hint\textsuperscript{794} of this.
2. If state is default\textsuperscript{793}, then return the empty string.
3. If state is none\textsuperscript{793}, then return "none"\textsuperscript{794}.
4. If state is sentences\textsuperscript{793}, then return "sentences"\textsuperscript{794}.
5. Return the keyword value corresponding to state.

The autocapitalize\textsuperscript{794} setter steps are to set the autocapitalize\textsuperscript{794} content attribute to the given value.

User agents that support customizable autocapitalization behavior for a text input method and wish to allow web developers to control this functionality should, during text input into an element, compute the used autocapitalization hint for the element. This will be an autocapitalization hint\textsuperscript{794} that describes the recommended autocapitalization behavior for text input into the element.

User agents or input methods may choose to ignore or override the used autocapitalization hint\textsuperscript{794} in certain circumstances.

The used autocapitalization hint\textsuperscript{794}, for an element element is computed using the following algorithm:

1. If element is an input\textsuperscript{493} element whose type\textsuperscript{495} attribute is in one of the URL\textsuperscript{501}, Email\textsuperscript{502}, or Password\textsuperscript{503} states, then return default\textsuperscript{793}.
2. If element is an input\textsuperscript{493} element or a textarea\textsuperscript{548} element, then return element's own autocapitalization hint\textsuperscript{794}.
3. If element is an editing host\textsuperscript{791} or an editable element, then return the own autocapitalization hint\textsuperscript{794} of the editing host of element.
4. Assert: this step is never reached, since text input only occurs in elements that meet one of the above criteria.
6.7.7 Input modalities: the `inputmode` attribute

User agents can support the `inputmode` attribute on form controls (such as the value of `textarea` elements), or in elements in an editing host (e.g., using `contenteditable`).

The `inputmode` content attribute is an enumerated attribute that specifies what kind of input mechanism would be most helpful for users entering content.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>The user agent should not display a virtual keyboard. This keyword is useful for content that renders its own keyboard control.</td>
</tr>
<tr>
<td>text</td>
<td>The user agent should display a virtual keyboard capable of text input in the user's locale.</td>
</tr>
<tr>
<td>tel</td>
<td>The user agent should display a virtual keyboard capable of telephone number input. This should include keys for the digits 0 to 9, the &quot;#&quot; character, and the &quot;*&quot; character. In some locales, this can also include alphabetic mnemonic labels (e.g., in the US, the key labeled &quot;2&quot; is historically also labeled with the letters A, B, and C).</td>
</tr>
<tr>
<td>url</td>
<td>The user agent should display a virtual keyboard capable of text input in the user's locale, with keys for aiding in the input of URLs, such as that for the &quot;/&quot; and &quot;.&quot; characters and for quick input of strings commonly found in domain names such as &quot;www.&quot; or &quot;*.com&quot;.</td>
</tr>
<tr>
<td>email</td>
<td>The user agent should display a virtual keyboard capable of text input in the user's locale, with keys for aiding in the input of email addresses, such as that for the &quot;@&quot; character and the &quot;.&quot; character.</td>
</tr>
<tr>
<td>numeric</td>
<td>The user agent should display a virtual keyboard capable of numeric input. This keyword is useful for PIN entry.</td>
</tr>
<tr>
<td>decimal</td>
<td>The user agent should display a virtual keyboard capable of fractional numeric input. Numeric keys and the format separator for the locale should be shown.</td>
</tr>
<tr>
<td>search</td>
<td>The user agent should display a virtual keyboard optimized for search.</td>
</tr>
</tbody>
</table>

The `inputMode` IDL attribute must reflect the `inputmode` content attribute, limited to only known values.

When `inputmode` is unspecified (or is in a state not supported by the user agent), the user agent should determine the default virtual keyboard to be shown. Contextual information such as the input `type` or `pattern` attributes should be used to determine which type of virtual keyboard should be presented to the user.

6.7.8 Input modalities: the `enterkeyhint` attribute

User agents can support the `enterkeyhint` attribute on form controls (such as the value of `textarea` elements), or in elements in an editing host (e.g., using `contenteditable`).

The `enterkeyhint` content attribute is an enumerated attribute that specifies what action label (or icon) to present for the enter key on virtual keyboards. This allows authors to customize the presentation of the enter key in order to make it more helpful for users.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enter</td>
<td>The user agent should present a cue for the operation 'enter', typically inserting a new line.</td>
</tr>
<tr>
<td>done</td>
<td>The user agent should present a cue for the operation 'done', typically meaning there is nothing more to input and the input method editor (IME) will be closed.</td>
</tr>
<tr>
<td>go</td>
<td>The user agent should present a cue for the operation 'go', typically meaning to take the user to the target of the text they typed.</td>
</tr>
<tr>
<td>next</td>
<td>The user agent should present a cue for the operation 'next', typically taking the user to the next field that will accept text.</td>
</tr>
<tr>
<td>previous</td>
<td>The user agent should present a cue for the operation 'previous', typically taking the user to the previous field that will accept text.</td>
</tr>
<tr>
<td>search</td>
<td>The user agent should present a cue for the operation 'search', typically taking the user to the results of searching for the text they have typed.</td>
</tr>
<tr>
<td>send</td>
<td>The user agent should present a cue for the operation 'send', typically delivering the text to its target.</td>
</tr>
</tbody>
</table>

The `enterKeyHint` IDL attribute must reflect the `enterkeyhint` content attribute, limited to only known values.

When `enterkeyhint` is unspecified (or is in a state not supported by the user agent), the user agent should determine the default action label (or icon) to present. Contextual information such as the `inputmode`, `type`, or `pattern` attributes should be used to determine which action label (or icon) to present on the virtual keyboard.

6.8 Find-in-page

6.8.1 Introduction

This section defines find-in-page — a common user-agent mechanism which allows users to search through the contents of the page.
for particular information.

Access to find-in-page feature is provided via a find-in-page interface. This is a user-agent provided user interface, which allows the user to specify input and the parameters of the search. This interface can appear as a result of a shortcut or a menu selection.

A combination of text input and settings in the find-in-page interface represents the user query. This typically includes the text that the user wants to search for, as well as optional settings (e.g., the ability to restrict the search to whole words only).

The user-agent processes page contents for a given query, and identifies zero or more matches, which are content ranges that satisfy the user query.

One of the matches is identified to the user as the active match. It is highlighted and scrolled into view. The user can navigate through the matches by advancing the active match using the find-in-page interface.

Issue #3539 tracks standardizing how find-in-page underlies the currently-unspecified window.find() API.

6.8.2 Interaction with selection

The find-in-page process is invoked in the context of a document, and may have an effect on the selection of that document. Specifically, the range that defines the active match can dictate the current selection. These selection updates, however, can happen at different times during the find-in-page process (e.g. upon the find-in-page interface dismissal or upon a change in the active match range).

6.9 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 they cycle through possibilities), 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).

6.9.1 Introduction

This section is non-normative.

To make an element draggable, 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:

```html
<p>What fruits do you like?</p>
```
<ol ondragstart="dragStartHandler(event)"
    <li draggable="true" data-value="fruit-apple">Apples</li>
    <li draggable="true" data-value="fruit-orange">Oranges</li>
    <li draggable="true" data-value="fruit-pear">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 data-value="" attribute as the value to be moving:
      event.dataTransfer.setData(internalDNDType, event.target.dataset.value);
      event.dataTransfer.effectAllowed = 'move'; // only allow moves
    } else {
      event.preventDefault(); // don't allow selection to be dragged
    }
  }
</script>

To accept a drop, the drop target has to listen to the following events:

1. The `dragenter` event handler reports whether or not the drop target is potentially willing to accept the drop, by canceling the event.

2. The `dragover` event handler specifies what feedback will be shown to the user, by setting the `dropEffect` attribute of the `DataTransfer` associated with the event. This event also needs to be canceled.

3. The `drop` event handler has a final chance to accept or reject the drop. If the drop is accepted, the event handler must perform the drop operation on the target. This event needs to be canceled, so that the `dropEffect` attribute's value can be used by the source. Otherwise, the drop operation is rejected.

For example:

<p>Drop your favorite fruits below:</p>

<ol ondragenter="dragEnterHandler(event)" ondragover="dragOverHandler(event)" ondrop="dropHandler(event)"
</ol>

<script>
  var internalDNDType = 'text/x-example'; // set this to something specific to your site
  function dragEnterHandler(event) {
    var items = event.dataTransfer.items;
    for (var i = 0; i < items.length; ++i) {
      var item = items[i];
      if (item.kind == 'string' && item.type == internalDNDType) {
        event.preventDefault();
        return;
      }
    }
  }
  function dragOverHandler(event) {
    event.dataTransfer.dropEffect = 'move';
    event.preventDefault();
  }
  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';
    }
  }
</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:

```html
<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>
```

### 6.9.2 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:
    
    - **Text**
      
      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.

    There is a limit of one text 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...
• 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 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".

### 6.9.3 The `DataTransfer` interface

`DataTransfer` objects are used to expose the drag data store that underlies a drag-and-drop operation.

```idl
[Exposed=Window]
interface DataTransfer {
  constructor();

  attribute DOMString-dropEffect;
  attribute DOMString-effectAllowed;

  [SameObject] readonly attribute DataTransferItemList items;

  undefined setDragImage(Element image, long x, long y);

  /* old interface */
  readonly attribute FrozenArray<DOMString> types;
  DOMString getData(DOMString format);
  undefined setData(DOMString format, DOMString data);
  undefined clearData(optional DOMString format);
  [SameObject] readonly attribute FileList files;
};
```

**For web developers (non-normative)**

`dataTransfer = new DataTransfer()`

Creates a new `DataTransfer` object with an empty drag data store.

`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 a `frozen 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 that are created as part of drag-and-drop events are only valid while those events are being fired.

A `DataTransfer` object is associated with a `drag data store` while it is valid.

A `DataTransfer` object has an associated `types array`, which is a `FrozenArray<DOMString>`, initially empty. When the contents of the `DataTransfer` object's `drag data store item list` change, or when the `DataTransfer` object becomes no longer associated with a `drag data store`, run the following steps:

1. Let `L` be an empty sequence.
2. If the `DataTransfer` object is still associated with a `drag data store`, then:
   1. For each item in the `DataTransfer` object's `drag data store item list` whose `kind` is text, add an entry to `L` consisting of the item's `type string`.
   2. If there are any items in the `DataTransfer` object's `drag data store item list` whose `kind` is `File`, then add an entry to `L` consisting of the string "Files". (This value can be distinguished from the other values because it is not lowercase.)
3. Set the `DataTransfer` object's `types array` to the result of creating a `frozen array` from `L`.

The `DataTransfer()` constructor, when invoked, must return a newly created `DataTransfer` object initialized as follows:

1. Set the `drag data store`'s `item list` to be an empty list.
2. Set the `drag data store`'s `mode` to `read/write mode`.
3. Set the `dropEffect` and `effectAllowed` to "none".

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 the `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, return. Nothing happens.
2. If the drag data store's mode is not the read/write mode, return. Nothing happens.
3. If element 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 this `DataTransfer` object's `types` array.

The `getData(format)` method must run the following steps:

1. If the `DataTransfer` object is no longer associated with a drag data store, then return the empty string.
2. If the drag data store's mode is the protected mode, then return the empty string.
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 text and whose type string is equal to `format`, return the empty string.
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`, if there is one.
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, return. Nothing happens.
2. If the drag data store's mode is not the read/write mode, return. 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 text 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 text, 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, return. Nothing happens.
2. If the drag data store's mode is not the read/write mode, return. 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 return.
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 text 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. Return the empty list `L`.
3. If the drag data store's mode is the protected mode, 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.

### 6.9.3.1 The `DataTransferItemList` interface

Each `DataTransfer` object is associated with a `DataTransferItemList` object.

**IDL**

```idl
[Exposed=Window]
interface DataTransferItemList {
  readonly attribute unsigned long length;
  getter DataTransferItem (unsigned long index);
  DataTransferItem? add(DOMString data, DOMString type);
  DataTransferItem? add(File data);
  undefined remove(unsigned long index);
  undefined clear();
};
```

**For web developers (non-normative)**

- `items.length`
  
  Returns the number of items in the drag data store.

- `items[index]`
  
  Returns the `DataTransferItem` object representing the index entry in the drag data store.

- `items.remove(index)`
  
  Removes the index entry in the drag data store.

- `items.clear()`
  
  Removes all the entries in the drag data store.

- `items.add(data)`
  
  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.
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 text and whose type string is equal to the value of the method's second argument, converted to ASCII lowercase, then throw a "NotSupportedException" DOMException.
     Otherwise, add an item to the drag data store item list whose kind is text, 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" DOMException.
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.

6.9.3.2 The DataTransferItem interface

Each DataTransferItem object is associated with a DataTransfer object.

```
IDL [Exposed=Window]
interface DataTransferItem {
    readonly attribute DOMString kind;
    readonly attribute DOMString type;
    undefined getAsString(FunctionStringCallback? _callback);
    File? getAsFile();
};

callback FunctionStringCallback = undefined (DOMString data);
```

For web developers (non-normative)

item.kind

Returns the drag data item kind, one of: "string", "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:

<table>
<thead>
<tr>
<th>Kind</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>&quot;string&quot;</td>
</tr>
<tr>
<td>File</td>
<td>&quot;file&quot;</td>
</tr>
</tbody>
</table>

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, return.
2. If the `DataTransferItem` object is not in the read/write mode or the read-only mode, return. The callback is never invoked.
3. If the drag data item kind is not text, then return. 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, then return null.
2. If the drag data item kind is not `File`, then return null.
3. Return a new `File` object representing the actual data of the item represented by the `DataTransferItem` object.

### 6.9.4 The `DragEvent` interface

The drag-and-drop processing model involves several events. They all use the `DragEvent` interface.

```idl
[Exposed=Window]
interface DragEvent : MouseEvent {
    constructor(DOMString type, optional DragEventInit eventInitDict = {});

    readonly attribute DataTransfer? dataTransfer;
};
```

```dictionary
DragEventInit : MouseEventInit {
```
For web developers (non-normative)

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. Let dataDragStoreWasChanged be false.
2. If no specific related target was provided, set related target to null.
3. Let window be the relevant global object of the Document object of the specified target element.
4. If e is dragstart, then set the drag data store to the read/write mode and set dataDragStoreWasChanged to true.
   If e is drop, set the drag data store to the read-only mode.
5. Let dataTransfer be a newly created DataTransfer object associated with the given drag data store.
6. Set the effectAllowed attribute to the drag data store's drag data store allowed effects state.
7. Set the dropEffect attribute to "none" if e is dragstart, drag, 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):

<table>
<thead>
<tr>
<th>effectAllowed</th>
<th>dropEffect</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;none&quot;</td>
<td>&quot;none&quot;</td>
</tr>
<tr>
<td>&quot;copy&quot;</td>
<td>&quot;copy&quot;</td>
</tr>
<tr>
<td>&quot;copyLink&quot;</td>
<td>&quot;copy&quot;</td>
</tr>
<tr>
<td>&quot;copyMove&quot;</td>
<td>&quot;copy&quot;</td>
</tr>
<tr>
<td>&quot;all&quot;</td>
<td>&quot;copy&quot;</td>
</tr>
<tr>
<td>&quot;link&quot;</td>
<td>&quot;link&quot;</td>
</tr>
<tr>
<td>&quot;linkMove&quot;</td>
<td>&quot;link&quot;</td>
</tr>
<tr>
<td>&quot;move&quot;</td>
<td>&quot;move&quot;</td>
</tr>
<tr>
<td>&quot;uninitialized&quot; when what is being dragged is a selection from a text control</td>
<td>&quot;move&quot;</td>
</tr>
<tr>
<td>&quot;uninitialized&quot; when what is being dragged is a selection</td>
<td>&quot;move&quot;</td>
</tr>
<tr>
<td>&quot;uninitialized&quot; when what is being dragged is an element with an attribute</td>
<td>&quot;move&quot;</td>
</tr>
</tbody>
</table>

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

For example, Windows platform conventions are such that dragging while holding the "alt" key indicates a preference for...
8. Let event be the result of creating an event using DragEvent.

9. Initialize event's type attribute to e, its bubbles attribute to true, its view attribute to window, its relatedTarget attribute to related target, and its dataTransfer attribute to dataTransfer.

10. If e is not dragleave or drageend, then initialize event's cancelable attribute to true.

11. Initialize event's mouse and key attributes initialized according to the state of the input devices as they would be for user interaction events.

   If there is no relevant pointing device, then initialize event's screenX, screenY, clientX, clientY, and button attributes to 0.

12. Dispatch event at the specified target element.

13. 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.)

14. If dataDragStoreWasChanged is true, then set the drag data store back to the protected mode.

15. Break the association between dataTransfer and the drag data store.

6.9.5 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 set to true. If there is no such element, then nothing is being dragged; return, 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.

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.

   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**
   
   Text

   **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**
   
   Text

   **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 element with an attribute**
      
      Add to urls the result of parsing the element's content attribute relative to the element's document.

      **If the node is an element with a src attribute**
      
      Add to urls the result of parsing the element's src content attribute relative to the element's document.
3. If urls is still empty, then return.

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
   Text

   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; return.

   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]

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 (±200ms) 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, return 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 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.

   Otherwise, run the appropriate step from the following list:

   - If the immediate user selection is a text control (e.g., textarea, or an input element whose type attribute is in the Text, 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 text
     Set the current target element to the immediate user selection anyway.

   - 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.

2. 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.

3. 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 control (e.g., textarea, or an input element whose type attribute is in the Text, 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
     Set the current drag operation to either "copy" or "move", as appropriate given the platform conventions.

   - 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:

<table>
<thead>
<tr>
<th>effectAllowed</th>
<th>dropEffect</th>
<th>Drag operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;uninitialized&quot;, &quot;copy&quot;, &quot;copylink&quot;, &quot;copyMove&quot;, or &quot;all&quot;</td>
<td>&quot;copy&quot;</td>
<td>&quot;copy&quot;</td>
</tr>
<tr>
<td>&quot;uninitialized&quot;, &quot;link&quot;, &quot;copyLink&quot;, &quot;linkMove&quot;, or &quot;all&quot;</td>
<td>&quot;link&quot;</td>
<td>&quot;link&quot;</td>
</tr>
<tr>
<td>&quot;uninitialized&quot;, &quot;move&quot;, &quot;copyMove&quot;, &quot;linkMove&quot;, or &quot;all&quot;</td>
<td>&quot;move&quot;</td>
<td>&quot;move&quot;</td>
</tr>
<tr>
<td>Any other case</td>
<td>&quot;none&quot;</td>
<td>&quot;none&quot;</td>
</tr>
</tbody>
</table>

4. 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.

5. Update the drag feedback (e.g. the mouse cursor) to match the current drag operation, as follows:

<table>
<thead>
<tr>
<th>Drag operation</th>
<th>Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>copy</strong></td>
<td>Data will be copied if dropped here.</td>
</tr>
<tr>
<td><strong>link</strong></td>
<td>Data will be linked if dropped here.</td>
</tr>
<tr>
<td><strong>move</strong></td>
<td>Data will be moved if dropped here.</td>
</tr>
<tr>
<td><strong>none</strong></td>
<td>No operation allowed, dropping here will cancel the drag-and-drop operation.</td>
</tr>
</tbody>
</table>

4. 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:

   1. 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:

      1. Let `dropped` be false.
      2. 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 control (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 and the drag data item kind text:

      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 text into the text control 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 control (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 control (see below), the current drag operation is "move", and the source of the drag-and-drop operation is a selection in a text control:

      The user agent should delete the dragged selection from the relevant text control.
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 control is a textarea element or an input element whose type attribute is in one of the Text, Search, Tel, URL, Email, 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.

6.9.6 Events summary

This section is non-normative.

The following events are involved in the drag-and-drop model.

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Target</th>
<th>Cancelable?</th>
<th>Drag data store mode</th>
<th>dropEffect</th>
<th>Default Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>dragstart</td>
<td>Source node</td>
<td>✓ Cancelable</td>
<td>Read/write mode</td>
<td>&quot;none&quot;</td>
<td>Initiate the drag-and-drop operation</td>
</tr>
<tr>
<td>drag</td>
<td>Source node</td>
<td>✓ Cancelable</td>
<td>Protected mode</td>
<td>&quot;none&quot;</td>
<td>Continue the drag-and-drop operation</td>
</tr>
<tr>
<td>dragenter</td>
<td>Immediate user selection or the body element</td>
<td>✓ Cancelable</td>
<td>Protected mode</td>
<td>Based on effectAllowed value</td>
<td>Reject immediate user selection as potential target element</td>
</tr>
<tr>
<td>dragleave</td>
<td>Previous target element</td>
<td>—</td>
<td>Protected mode</td>
<td>&quot;none&quot;</td>
<td>None</td>
</tr>
<tr>
<td>dragover</td>
<td>Current target element</td>
<td>✓ Cancelable</td>
<td>Protected mode</td>
<td>Based on effectAllowed value</td>
<td>Reset the current drag operation to &quot;none&quot;</td>
</tr>
<tr>
<td>drop</td>
<td>Current target element</td>
<td>✓ Cancelable</td>
<td>Read-only mode</td>
<td>Current drag operation</td>
<td>Varies</td>
</tr>
<tr>
<td>dragend</td>
<td>Source node</td>
<td>✓ Cancelable</td>
<td>Protected mode</td>
<td>Current drag operation</td>
<td>Varies</td>
</tr>
</tbody>
</table>

Not shown in the above table: all these events bubble, are composed, and the effectAllowed attribute always has the value it had after the dragstart event, defaulting to "uninitialized" in the dragstart event.

6.9.7 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 and the invalid 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.
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".

6.9.8 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 drop 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 their mouse button depressed, the UA would not consider that to start a drag. This is important because otherwise UAs 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

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.
7 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.

7.1 Browsing contexts

A browsing context is an environment in which Document objects are presented to the user.

A browsing context has a corresponding WindowProxy object.

A browsing context has an opener browsing context, which is null or a browsing context. It is initially null.

A browsing context has a disowned boolean. It is initially false.

A browsing context has an is closing boolean. It is initially false.

A browsing context's active window is its WindowProxy object's [[Window]] internal slot value. A browsing context's active document is its active window's associated Document.

Note

A tab or window in a web browser typically contains a browsing context, as does an iframe or frame in a frameset.

A browsing context has a corresponding WindowProxy object.

A browsing context has an opener browsing context, which is null or a browsing context. It is initially null.

A browsing context has a disowned boolean. It is initially false.

A browsing context has an is closing boolean. It is initially false.

Example

The following example illustrates the various possibilities of a browsing context. It can be disowned, is closing, neither, or both.

```javascript
// Neither disowned nor is closing:
const popup1 = window.open();

// Disowned, but not is closing:
const popup2 = window.open();
popup2.opener = null;

// Not disowned, but is closing:
const popup3 = window.open();
popup3.close();

// Disowned, is closing:
const popup4 = window.open();
popup4.opener = null;
popup4.close();
```

A browsing context has a session history, which lists the Document objects that the browsing context has presented, is presenting, or will present. A Document's browsing context is the browsing context whose session history contains the Document, if any such browsing context exists and has not been discarded, and null otherwise.

Note

A Document does not necessarily have a non-null browsing context. In particular, data mining tools are likely to never instantiate browsing contexts. A Document created using an API such as createDocument() never has a non-null browsing context. And the Document originally created for an iframe element, which has since been removed from the document, has no associated browsing context, since that browsing context was discarded.

A browsing context's active window is its WindowProxy object's [[Window]] internal slot value. A browsing context's active document is its active window's associated Document.
A browsing context has a virtual browsing context group ID integer. It is initially 0. This is used by cross-origin opener policy reporting, to keep track of the browsing context group switches that would have happened if the report-only policy had been enforced.

A browsing context has an initial URL, which is a URL or null. It is initially null.

A browsing context has an opener origin at creation, which is an origin or null. It is initially null.

To set the active document of a browsing context browsingContext to a Document object document, run these steps:

1. Let window be document's relevant global object.
2. Set browsingContext's active window to window.
4. Set window's relevant settings object's execution ready flag.

A browsing context has an associated creator origin (null or returns an origin), creator URL (null or returns a URL), and creator base URL (null or returns a URL). These are all initially null.

To determine the origin, given browsing context browsingContext, URL url, sandboxing flag set sandboxFlags, and two origins invocationOrigin and activeDocumentNavigationOrigin:

1. If sandboxFlags has its sandboxed origin browsing context flag set, then return a new opaque origin.
2. If url is null, then return a new opaque origin.
3. If activeDocumentNavigationOrigin is not null, and url's scheme is "javascript", then return activeDocumentNavigationOrigin.
4. If invocationOrigin is non-null and url is about:blank, then return invocationOrigin.
5. If url is about:srcdoc, then return the origin of browsingContext's container document.
6. Return url's origin.

To create a new browsing context, given null or a Document object creator, null or an element embedder, and a browsing context group group, run these steps:

1. Let browsingContext be a new browsing context.
2. If creator is non-null, then set browsingContext's creator origin to return creator's origin, browsingContext's creator URL to return creator's URL, browsingContext's creator base URL to return creator's base URL, and browsingContext's virtual browsing context group ID to creator's top-level browsing context's virtual browsing context group ID.
3. Let sandboxFlags be the result of determining the creation sandboxing flags given browsingContext and embedder.
4. Let origin be the result of determining the origin given browsingContext, about:blank, sandboxFlags,
5. Let permissionsPolicy be the result of creating a permissions policy given browsingContext and origin.

   This needs to use embedder.

6. Let agent be the result of obtaining a similar-origin window agent given origin, group, and false.

7. Let realm execution context be the result of creating a new JavaScript realm given agent and the following customizations:
   - For the global object, create a new Window object.
   - For the global this binding, use browsingContext's WindowProxy object.

8. Let topLevelCreationURL be about:blank if embedder is null; otherwise embedder's relevant settings object's top-level creation URL.

9. Let topLevelOrigin be origin if embedder is null; otherwise embedder's relevant settings object's top-level origin.

10. Set up a window environment settings object with about:blank, realm execution context, null, topLevelCreationURL, and topLevelOrigin.

11. Let coop be a new cross-origin opener policy.

12. If creator is non-null and creator's origin is same origin with creator's relevant settings object's top-level origin, then set coop to creator's browsing context's top-level browsing context's active document's cross-origin opener policy.

13. Let document be a new Document, marked as an HTML document in quirks mode, whose content type is "text/html", origin is origin, active sandboxing flag set is sandboxFlags, permissions policy is permissionsPolicy, cross-origin opener policy is coop, and which is ready for post-load tasks.


15. Ensure that document has a single child html node, which itself has two empty child nodes: a head element, and a body element.

16. Set the active document of browsingContext to document.

17. If browsingContext's creator URL is non-null, then set document's referrer to the serialization of it.

18. If creator is non-null, then set document's referrer policy to creator's referrer policy.

19. If creator is non-null, then set document's embedder policy to creator's embedder policy.

20. Add document to browsingContext's session history.


22. Return browsingContext.

To create a new top-level browsing context:

1. Let group be the result of creating a new browsing context group.

2. Return group's browsing context set[0].

Note

This creates a top-level browsing context.

To create a new auxiliary browsing context, given a browsing context opener:

1. Let group be opener's top-level browsing context's group.

2. Assert: group is non-null, as navigating invokes this directly.
3. Let browsingContext be the result of creating a new browsing context with opener's active document, null, and group.

4. Append browsingContext to group.

5. Set browsingContext's opener browsing context to opener.

6. Set browsingContext's virtual browsing context group ID to opener's top-level browsing context's virtual browsing context group ID.

7. Set browsingContext's opener origin at creation to opener's active document's origin.

8. Legacy-clone a browsing session storage shed with opener's browsing session and browsingContext's browsing session. [STORAGE]


**Note**

This creates a top-level browsing context that is also an auxiliary browsing context.

To create a new nested browsing context, given an element:

1. Let group be element's node document's browsing context's top-level browsing context's group.

2. Let browsingContext be the result of creating a new browsing context with element's node document, element, and group.

3. Set element's nested browsing context to browsingContext.

4. If element has a name attribute, then set browsingContext's name to the value of this attribute.

7.1.2 Related browsing contexts

Certain elements (for example, iframe elements) can instantiate further browsing contexts. These elements are called browsing context containers.

Each browsing context container has a nested browsing context, which is either a browsing context or null. It is initially null.

The container of a browsing context bc is the browsing context container whose nested browsing context is bc, or null if there is no such element.

Each browsing context bc has a container document, which is the result of running these steps:

1. If bc's container is null, then return null.

2. Return bc's container's node document.

**Note**

This is equal to bc's container's shadow-including root as bc's container has to be connected.

A browsing context child is said to be a child browsing context of another browsing context parent, if child's container is non-null and child's container's browsing context is parent.

A browsing context child is a document-tree child browsing context of parent if child is a child browsing context and child's container is in a document tree.

A browsing context child may have a parent browsing context. This is the unique browsing context that has child as a child browsing context, if any such browsing context exists. Otherwise, the browsing context 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 has no parent browsing context is the top-level browsing context for itself and all of the browsing contexts for which it is an ancestor browsing context.
A top-level browsing context has an associated group (null or a browsing context group). It is initially null.

It is possible to create new browsing contexts that are related to a top-level browsing context while their container is null. Such browsing contexts are called auxiliary browsing contexts. Auxiliary browsing contexts are always top-level browsing contexts.

The transitive closure of parent browsing contexts for a browsing context that is a child browsing context gives the list of ancestor browsing contexts.

The list of the descendant browsing contexts of a Document is the (ordered) list returned by the following algorithm:

1. Let list be an empty list.
2. For each browsing context container container, whose nested browsing context is non-null and whose shadow-including root is d, in shadow-including tree order:
   1. Let nestedBC be container's nested browsing context.
   2. Append nestedBC to list.
   3. Extend list with the list of the descendant browsing contexts of nestedBC's active document.
3. Return list.

A Document is said to be fully active when d's browsing context is non-null, d's browsing context's active document is d, and either d's browsing context is a top-level browsing context, or d's browsing context's container document is fully active.

Because they are associated with 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 Document is said to be fully active when d's browsing context is non-null, d's browsing context's active document is d, and either d's browsing context is a top-level browsing context, or d's browsing context's container document is fully active.

Because they are associated with 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.

Example

The following example illustrates the differences between active and fully active Document objects. Here a.html is loaded into a browser window, b-1.html starts out loaded into an iframe as shown, and b-2.html and c.html are omitted (they can simply be an empty document).

```
<!DOCTYPE html>
<html lang="en">
<title>Browsing context A</title>

<iframe src="b-1.html"></iframe>
<button onclick="frames[0].location.href = 'b-2.html'">Click me</button>

<!DOCTYPE html>
<html lang="en">
<title>Browsing context B</title>

<iframe src="c.html"></iframe>
```

At this point, the documents given by a.html, b-1.html, and c.html are all the active documents of their respective browsing contexts. They are also all fully active.

After clicking on the button, and thus loading a new Document from b-2.html into browsing context B, we have the following results:

- The a.html Document remains both the active of browsing context A, and fully active.
- The b-1.html Document is now not the active of browsing context B. As such it is also not fully active.
- The new b-2.html Document is now the active of browsing context B, and is also fully active.
7.1.2.1 Navigating related browsing contexts in the DOM

For web developers (non-normative)

- **window.top**
  - Returns the `WindowProxy` for the top-level browsing context.

- **window.opener**
  - Returns the `WindowProxy` for the opener browsing context.
  - Returns null if there isn't one or if it has been set to null.
  - Can be set to null.

- **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 attribute's getter must run these steps:

1. If this Window object's browsing context is null, then return null.
2. Return this Window object's browsing context's top-level browsing context's `WindowProxy` object.

The opener attribute's getter must run these steps:

1. Let current be this Window object's browsing context.
2. If current is null, then return null.
3. If current's disowned is true, then return null.
4. If current's opener browsing context is null, then return null.
5. Return current's opener browsing context's WindowProxy object.

The opener attribute's setter must run these steps:

1. If the given value is null and this Window object's browsing context is non-null, then set this Window object's browsing context's disowned to true.
2. If the given value is non-null, then return \texttt{OrdinaryDefineOwnProperty} (this Window object, "opener", \{ [[Value]]: the given value, [[Writable]]: true, [[Enumerable]]: true, [[Configurable]]: true \}).

\textbf{Note}

If a browsing context's disowned is true, its window.opener attribute is null. That prevents scripts in the browsing context from changing any properties of its opener browsing context's Window object (i.e., the Window object from which the browsing context was created).

Otherwise, if a browsing context's disowned is false, then scripts in that browsing context can use window.opener to change properties of its opener browsing context's Window object. For example, a script running in the browsing context can change the value of window.opener.location, causing the opener browsing context to navigate to a completely different document.

The parent attribute's getter must run these steps:

1. Let current be this Window object's browsing context.
2. If current is null, then return null.
3. If current is a child browsing context of another browsing context parent, then return parent's WindowProxy object.
4. Assert: current is a top-level browsing context.
5. Return current's WindowProxy object.

The frameElement attribute's getter must run these steps:

1. Let current be this Window object's browsing context.
2. If current is null, then return null.
3. Let container be current's container.
4. If container is null, then return null.
5. If container's node document's origin is not same origin-domain with the current settings object's origin, then return null.
6. Return container.

\textbf{Example}

An example of when these IDL attributes can return null is as follows:

```
<!DOCTYPE html>
<iframe></iframe>

<script>
"use strict";
const element = document.querySelector("iframe");
const iframeWindow = element.contentWindow;
element.remove();

console.assert(iframeWindow.top === null);
console.assert(iframeWindow.parent === null);
console.assert(iframeWindow.frameElement === null);
</script>
```
Here the browsing context\textsuperscript{813} corresponding to iframeWindow was discarded\textsuperscript{834} when element was removed from the document.

### 7.1.3 Security \textsuperscript{822}

A browsing context\textsuperscript{811} A is familiar with a second browsing context\textsuperscript{811} B if one of the following conditions is true:

- Either the origin of the active document\textsuperscript{813} of A is the same\textsuperscript{841} as the origin of the active document\textsuperscript{811} of B, or
- The browsing context A is a child browsing context\textsuperscript{816} and its top-level browsing context\textsuperscript{816} is B, or
- The browsing context B is an auxiliary browsing context\textsuperscript{817} and A is familiar with\textsuperscript{820} B's opener browsing context\textsuperscript{813}, or
- The browsing context B is not a top-level browsing context\textsuperscript{816}, but there exists an ancestor browsing context\textsuperscript{816} of B whose active document\textsuperscript{811} has the same\textsuperscript{841} origin as the active document\textsuperscript{811} of A (possibly in fact being A itself).

A browsing context\textsuperscript{811} A is allowed to navigate a second browsing context\textsuperscript{811} B if the following algorithm returns true:

1. If A is not the same browsing context\textsuperscript{811} as B, and A is not one of the ancestor browsing contexts\textsuperscript{816} of B, and B is not a top-level browsing context\textsuperscript{816}, and A's active document\textsuperscript{811}'s active sandboxing flag set\textsuperscript{847} has its sandboxed navigation browsing context flag\textsuperscript{844} set, then return false.

2. Otherwise, if B is a top-level browsing context\textsuperscript{816}, and is one of the ancestor browsing contexts\textsuperscript{816} of A, then:
   
   1. If A's active window\textsuperscript{811} has transient activation\textsuperscript{769} and A's active document\textsuperscript{811}'s active sandboxing flag set\textsuperscript{847} has its sandboxed top-level navigation with user activation browsing context flag\textsuperscript{844} set, then return false.

   2. Otherwise, if A's active window\textsuperscript{811} does not have transient activation\textsuperscript{769} and A's active document\textsuperscript{811}'s active sandboxing flag set\textsuperscript{847} has its sandboxed top-level navigation without user activation browsing context flag\textsuperscript{844} set, then return false.

3. Otherwise, if B is a top-level browsing context\textsuperscript{816}, and is neither A nor one of the ancestor browsing contexts\textsuperscript{816} of A, and A's Document\textsuperscript{811}'s active sandboxing flag set\textsuperscript{847} has its sandboxed navigation browsing context flag\textsuperscript{844} set, and A is not the one permitted sandboxed navigator\textsuperscript{844} of B, then return false.

4. Return true.

An element has a browsing context scope origin if its Document\textsuperscript{811}'s browsing context\textsuperscript{811} is a top-level browsing context\textsuperscript{816} or if all of its Document\textsuperscript{811}'s ancestor browsing contexts\textsuperscript{816} all have active documents\textsuperscript{813} whose origins are the same origin\textsuperscript{841} as the element's node document's origin. If an element has a browsing context scope origin\textsuperscript{820}, then its value is the origin of the element's node document.

### 7.1.4 Groupings of browsing contexts \textsuperscript{822}

A user agent holds a browsing context group set (a set of browsing context groups\textsuperscript{820}).

A browsing context group holds a browsing context set (a set of top-level browsing contexts\textsuperscript{816}).

**Note**

A top-level browsing context\textsuperscript{816} is added to the group\textsuperscript{820} when the group is created\textsuperscript{821}. All subsequent top-level browsing contexts\textsuperscript{816} added to the group\textsuperscript{820} will be auxiliary browsing contexts\textsuperscript{817}.

A browsing context group\textsuperscript{820} has an associated agent cluster map (a weak map of agent cluster keys\textsuperscript{860} to agent clusters). User agents are responsible for collecting agent clusters when it is deemed that nothing can access them anymore.

A browsing context group\textsuperscript{820} has an associated historical agent cluster key map, which is a map of origins\textsuperscript{840} to agent cluster keys\textsuperscript{860}. This map is used to ensure the consistency of the origin-keyed agent clusters\textsuperscript{843} feature by recording what agent cluster keys
were previously used for a given origin.

A **browsing context group** has a **cross-origin isolation mode**, which is a **cross-origin isolation mode**. It is initially "none".

A **cross-origin isolation mode** is one of three possible values: "none", "logical", or "concrete".

---

**Note**

The historical agent cluster key map only ever gains entries over the lifetime of the browsing context group.

"logical" and "concrete" are similar. They are both used for **browsing context groups** where:

- every top-level Document has `Cross-Origin-Opener-Policy: same-origin` and
- every Document has `Cross-Origin-Embedder-Policy: require-corp`.

On some platforms, it is difficult to provide the security properties required to grant safe access to the APIs gated by the **cross-origin isolated capability**. As a result, only "concrete" can grant access that capability. "logical" is used on platform not supporting this capability, where various restrictions imposed by cross-origin isolation will still apply, but the capability is not granted.

---

**To create a new browsing context group**, run these steps:

1. Let `group` be a new **browsing context group**.
2. Append `group` to the user agent's **browsing context group set**.
3. Let `browsingContext` be the result of **creating a new browsing context** with null, null, and `group`.
4. Append `browsingContext` to `group`.
5. Return `group`.

**To append** a **top-level browsing context** **browsingContext** to a **browsing context group** `group`, run these steps:

1. Append `browsingContext` to `group`'s **browsing context set**.
2. Set `browsingContext`'s `group` to `group`.

**To remove** a **top-level browsing context** **browsingContext**, run these steps:

1. Assert: `browsingContext`'s `group` is non-null, because a **browsing context** only gets **discarded** once.
2. Let `group` be `browsingContext`'s `group`.
3. Set `browsingContext`'s `group` to null.
4. Remove `browsingContext` from `group`'s **browsing context set**.
5. If `group`'s **browsing context set** is empty, then remove `group` from the user agent's **browsing context group set**.

---

**Note**

**Append** and **remove** are primitive operations that help define the lifetime of a **browsing context group**. They are called from **creating a new browsing context group**, **creating a new auxiliary browsing context**, and **discarding a browsing context**.

---

**Note**

The HTML Standard used to define "unit of related browsing contexts" and "unit of related similar-origin browsing contexts". These have been removed as they were not adequate.
7.1.5 Browsing context names

Browsing contexts can have a **browsing context name**. Unless stated otherwise, it is the empty string.

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, "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.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Ordinary effect</th>
<th>Effect in an <code>iframe</code> with…</th>
</tr>
</thead>
<tbody>
<tr>
<td>none specified, for links and form submissions</td>
<td>current</td>
<td>sandbox=&quot;&quot;</td>
</tr>
<tr>
<td>empty string</td>
<td>current</td>
<td>sandbox=&quot;allow-top-navigation&quot;</td>
</tr>
<tr>
<td><code>_blank</code></td>
<td>new</td>
<td>maybe new</td>
</tr>
<tr>
<td><code>_self</code></td>
<td>current</td>
<td>current</td>
</tr>
<tr>
<td><code>_parent</code> if there isn’t a parent</td>
<td>current</td>
<td>current</td>
</tr>
<tr>
<td><code>_parent</code> if parent is also top</td>
<td>parent/top</td>
<td>none</td>
</tr>
<tr>
<td><code>_parent</code> if there is one and it’s not top</td>
<td>parent</td>
<td>none</td>
</tr>
<tr>
<td><code>_top</code> if top is current</td>
<td>current</td>
<td>current</td>
</tr>
<tr>
<td><code>_top</code> if top is not current</td>
<td>top</td>
<td>none</td>
</tr>
<tr>
<td>name that doesn’t exist</td>
<td>new</td>
<td>maybe new</td>
</tr>
<tr>
<td>name that exists and is a descendant</td>
<td>specified descendant</td>
<td>specified descendant</td>
</tr>
<tr>
<td>name that exists and is current</td>
<td>current</td>
<td>current</td>
</tr>
<tr>
<td>name that exists and is an ancestor that is top</td>
<td>specified ancestor</td>
<td>none</td>
</tr>
<tr>
<td>name that exists and is an ancestor that is not top</td>
<td>specified ancestor</td>
<td>none</td>
</tr>
<tr>
<td>other name that exists with common top</td>
<td>specified</td>
<td>none</td>
</tr>
<tr>
<td>name that exists with different top, if familiar and one permitted sandboxed navigator</td>
<td>specified</td>
<td>specified</td>
</tr>
<tr>
<td>name that exists with different top, if familiar, but not one permitted sandboxed navigator</td>
<td>specified</td>
<td>none</td>
</tr>
<tr>
<td>name that exists with different top, not familiar</td>
<td>new</td>
<td>maybe new</td>
</tr>
</tbody>
</table>

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 below.

The rules for choosing a browsing context, given a browsing context name name, a browsing context current, and a boolean noopenner are as follows:

1. Let chosen be null.
2. Let windowType be "existing or none".
3. Let sandboxingFlagSet be current's active document's active sandboxing flag set.
4. If name is the empty string or an ASCII case-insensitive match for "_self", then set chosen to current.
5. Otherwise, if name is an ASCII case-insensitive match for "_parent", set chosen to current's parent browsing context, if any, and current otherwise.
6. Otherwise, if name is an ASCII case-insensitive match for "_top", set chosen to current's top-level browsing context, if any, and current otherwise.
7. Otherwise, if name is not an ASCII case-insensitive match for "_blank", there exists a browsing context whose name is
the same as name, current 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, set chosen to that browsing context. If there are multiple matching browsing contexts, the user agent should set chosen to one in some arbitrary consistent manner, such as the most recently opened, most recently focused, or more closely related.

This will be made more precise in issue #313.

8. 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 current’s active window does not have transient activation and the user agent has been configured to not show popups (i.e., the user agent has a "popup blocker" enabled)

   The user agent may inform the user that a popup has been blocked.

→ If sandboxingFlagSet has the sandboxed auxiliary navigation browsing context flag set

   The user agent may report to a developer console that a popup has been blocked.

→ If the user agent has been configured such that in this instance it will create a new browsing context

   1. Set windowType to "new and unrestricted".

   2. If current’s top-level browsing context’s cross-origin opener policy’s value is "same-origin" or "same-origin-plus-COEP", then:

      1. Let currentDocument be current’s active document.

      2. If currentDocument’s origin is not same origin with currentDocument’s relevant settings object’s top-level origin, then set no opener to true, name to "_blank", and windowType to "new with no opener".

   Note

   In the presence of a cross-origin opener policy, nested documents that are cross-origin with their top-level browsing context’s active document always set no opener to true.

   3. If no opener is true, then set chosen to the result of creating a new top-level browsing context.

   4. Otherwise:

      1. Set chosen to the result of creating a new auxiliary browsing context with current.

      2. If sandboxingFlagSet’s sandboxed navigation browsing context flag is set, then current must be set as chosen’s one permitted sandboxed navigator.

   5. If sandboxingFlagSet’s sandbox propagates to auxiliary browsing contexts flag is set, then all the flags that are set in sandboxingFlagSet must be set in chosen’s popup sandboxing flag set.

   6. If name is not an ASCII case-insensitive match for "_blank", then set chosen’s name to name.

   Note

   If the newly created browsing context is immediately navigated, then the navigation will be done with historyHandling set to "replace".

→ If the user agent has been configured such that in this instance it will reuse current

   Set chosen to current.

→ If the user agent has been configured such that in this instance it will not find a browsing context

   Do nothing.

   Note

   User agents are encouraged to provide a way for users to configure the user agent to always reuse current.

9. Return chosen and windowType.
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.

### 7.2 Security infrastructure for `Window`, `WindowProxy`, and `Location` objects

#### 7.2.1 Integration with IDL

When perform a security check is invoked, with a `platformObject`, `identifier`, and `type`, run these steps:

1. If `platformObject` is not a `Window` or `Location` object, then return.
2. For each `e` of `CrossOriginProperties`(`platformObject`):
   1. If `SameValue(e.[[Property]], identifier)` is true, then:
      1. If `type` is "method" and `e` has neither `[[NeedsGet]]` nor `[[NeedsSet]]`, 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.
3. If `IsPlatformObjectSameOrigin(``platformObject``)` is false, then throw a "SecurityError" DOMException.

#### 7.2.2 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 (currentGlobal, objectGlobal, propertyKey)-tuples and values are property descriptors, as a memoization of what is visible to scripts when currentGlobal inspects a `Window` or `Location` object from objectGlobal. 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**

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**

For example, setting `document.domain = "example.com"` 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.

#### 7.2.3 Shared abstract operations

**7.2.3.1 CrossOriginProperties (O)**

1. Assert: `O` is a `Location` or `Window` object.
2. If `O` is a `Location` object, then return « { [[Property]]: "href", [[NeedsGet]]: true, [[NeedsSet]]: false }, »

824

Note
Indexed properties do not need to be safelisted in this algorithm, as they are handled directly by the WindowProxy object.

A JavaScript property name P is a cross-origin accessible window property name if it is "window", "self", "location", "close", "closed", "focus", "blur", "frames", "length", "top", "opener", "parent", "postMessage", or an array index property name.

7.2.3.2 CrossOriginPropertyFallback ( P )
1. If P is "then", @@toStringTag, @@hasInstance, or @@isConcatSpreadable, then return PropertyDescriptor {{Value}: undefined, [[Writable]]: false, [[Enumerable]]: false, [[Configurable]]: true }.
2. Throw a "SecurityError" DOMException.

7.2.3.3 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.

Note
Here the current settings object roughly corresponds to the "caller", because this check occurs before the execution context for the getter/setter/method in question makes its way onto the JavaScript execution context stack. For example, in the code w.document, this step is invoked before the document getter is reached as part of the [[Get]] algorithm for the WindowProxy w.

7.2.3.4 CrossOriginGetOwnPropertyHelper ( O, P )
If this abstract operation returns undefined and there is no custom behavior, the caller needs to throw a "SecurityError" DOMException. In practice this is handled by the caller calling CrossOriginPropertyFallback.

1. Let crossOriginKey be a tuple consisting of the current settings object's origin, O's relevant settings object's origin, and P.
2. For each e of CrossOriginProperties ( O, P ):
   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 undefined.
      4. If e.[[NeedsGet]] and e.[[NeedsSet]] are absent, then:
         1. Let value be originalDesc.[[Value]].
         2. If ! IsCallable ( value ) is true, then set value to an anonymous built-in function, created in the current Realm Record, that performs the same steps as the IDL operation P on object O.
3. Set `crossOriginDesc` to `PropertyDescriptor`
   
   ```javascript
   { 
   [[Value]]: value, 
   [[Enumerable]]: false, 
   [[Writable]]: false, 
   [[Configurable]]: true 
   }.
   ```

5. Otherwise:

1. Let `crossOriginGet` be undefined.

2. If `e.[[NeedsGet]]` is true, then set `crossOriginGet` to an anonymous built-in function, created in the `current Realm Record`, that performs the same steps as the getter of the IDL attribute `P` on object `O`.

3. Let `crossOriginSet` be undefined.

4. If `e.[[NeedsSet]]` is true, then set `crossOriginSet` to an anonymous built-in function, created in the `current Realm Record`, that performs the same steps as the setter of the IDL attribute `P` on object `O`.

5. Set `crossOriginDesc` to `PropertyDescriptor`
   
   ```javascript
   { 
   [[Get]]: crossOriginGet, 
   [[Set]]: crossOriginSet, 
   [[Enumerable]]: false, 
   [[Configurable]]: true 
   }.
   ```

6. Create an entry in the value of the `[[CrossOriginPropertyDescriptorMap]]` internal slot of `O` with key `crossOriginKey` and value `crossOriginDesc`.

7. Return `crossOriginDesc`.

3. Return undefined.

---

**Note**

The reason that the property descriptors produced here are configurable is to preserve the invariants of the essential internal methods required by the JavaScript specification. In particular, since the value of the property can change as a consequence of navigation, it is required that the property be configurable. (However, see `tc39/ecma262 issue #672` and references to it elsewhere in this specification for cases where we are not able to preserve these invariants, for compatibility with existing web content.)

**Note**

The reason the property descriptors are non-enumerable, despite this mismatching the same-origin behavior, is for compatibility with existing web content. See `issue #3183` for details.

---

### 7.2.3.5 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, then throw a "`SecurityError`" `DOMException`.

7. Return `Call(getter, Receiver)`.

### 7.2.3.6 CrossOriginSet ( O, V, Receiver )

1. Let `desc` be `O.[[GetOwnProperty]]`(P).

2. Assert: `desc` is not undefined.

3. If `desc.[[Set]]` is present and its value is not undefined, then:
1. Perform `Call(setter, Receiver, «V»).

2. Return true.

4. Throw a "SecurityError" DOMException.

7.2.3.7 CrossOriginOwnPropertyKeys (O) §p82

1. Let keys be a new empty List.

2. For each e of !CrossOriginProperties(O), append e.[[Property]] to keys.

3. Return the concatenation of keys and « "then", @@toStringTag, @@hasInstance, @@isConcatSpreadable ».

7.3 The Window object §p82

```IDL
interface Window : EventTarget {
  // the current browsing context
  [LegacyUnforgeable] readonly attribute WindowProxy window;
  [Replaceable] readonly attribute WindowProxy self;
  [LegacyUnforgeable] readonly attribute Document document;
  attribute DOMString name;
  [PutForwards=href, LegacyUnforgeable] readonly attribute Location location;
  readonly attribute History history;
  readonly attribute CustomElementRegistry customElements;
  [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;
  undefined close();
  readonly attribute boolean closed;
  undefined stop();
  undefined focus();
  undefined blur();

  // other browsing contexts
  [Replaceable] readonly attribute WindowProxy frames;
  [Replaceable] readonly attribute unsigned long length;
  [LegacyUnforgeable] readonly attribute WindowProxy? top;
  attribute any opener;
  [Replaceable] readonly attribute WindowProxy? parent;
  readonly attribute Element? frameElement;
  WindowProxy? open(optional USVString url = "", optional DOMString target = "_blank", optional
  [LegacyNullToEmptyString] DOMString features = "");
  getter object (DOMString name);
  // Since this is the global object, the IDL named getter adds a NamedPropertiesObject exotic
  // object on the prototype chain. Indeed, this does not make the global object an exotic object.
  // Indexed access is taken care of by the WindowProxy exotic object.

  // the user agent
  readonly attribute Navigator navigator;

  Global=Window,
  Exposed=Window,
  LegacyUnenumerableNamedProperties]
```
readonly attribute boolean originAgentCluster;

// user prompts
undefined alert();
undefined alert(DOMString message);
boolean confirm(optional DOMString message = "");
DOMString? prompt(optional DOMString message = "", optional DOMString default = ");
undefined print();

undefined postMessage(any message, USVString targetOrigin, optional sequence<object> transfer = []);
undefined postMessage(any message, optional WindowPostMessageOptions options = {});

// also has obsolete members

Window includes GlobalEventHandlers;
Window includes WindowEventHandlers;

dictionary WindowPostMessageOptions : PostMessageOptions {
  USVString targetOrigin = "/";
}

For web developers (non-normative)

```javascript
window, window
window.frames
window.self
```

These attributes all return `window`.

```javascript
window, document
```

Returns the `Document` associated with `window`.

```javascript
document.defaultView
```

Returns the `Window` object of the `active document`.

Note

The `window` object has an **associated** `Document`, which is a `Document` object. It is set when the `window` object is created, and only ever changed during `navigation` from the initial `about:blank` `Document`.

The `window` object's **browsing context** is the `window` object's `associated Document`'s **browsing context**. **It is either null or a browsing context.**

The `window`, `frames`, and `self` attributes' getters must return this `window` object's **relevant Realm**'s `EnvironmentRecord`'s `[[GlobalThisValue]]`.

The `document` IDL attribute, on getting, must return this `window` object's **associated Document**.

Note

The `Document` object associated with a `window` object can change in exactly one case: when the `navigate` algorithm creates 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` attribute's getter, when invoked, must run these steps:

1. If this `Document` object's **browsing context** is null, then return null.
2. Return this `Document` object's **browsing context**'s `WindowProxy` object.

For historical reasons, `Window` objects must also have a writable, configurable, non-enumerable property named `HTMLDocument` whose value is the `Document` interface object.
7.3.1 APIs for creating and navigating browsing contexts by name

For web developers (non-normative)

```javascript
window = window.open([url [, target [, features ] ]])
```

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 `features` argument can be used to influence the rendering of the new window.

```javascript
window.name[ = value ]
```

Returns the name of the window.
Can be set, to change the name.

```javascript
window.close()
```

Closes the window.

```javascript
window.closed
```

Returns true if the window has been closed, false otherwise.

```javascript
window.stop()
```

Cancels the document load.

The **window open steps**, given a string `url`, a string `target`, and a string `features`, are as follows:

1. If the `event loop`'s `termination nesting level` is nonzero, return null.
2. Let `source browsing context` be the `entry global object`'s `browsing context`.
3. If `target` is the empty string, then set `target` to `"_blank"`.
4. Let `tokenizedFeatures` be the result of tokenizing `features`.
5. Let `noopener` and `noreferrer` be false.
6. If `tokenizedFeatures["noopener"]` exists, then:
   1. Set `noopener` to the result of parsing `tokenizedFeatures["noopener"]` as a boolean feature.
   2. Remove `tokenizedFeatures["noopener"]`.
7. If `tokenizedFeatures["noreferrer"]` exists, then:
   1. Set `noreferrer` to the result of parsing `tokenizedFeatures["noreferrer"]` as a boolean feature.
   2. Remove `tokenizedFeatures["noreferrer"]`.
8. If `noreferrer` is true, then set `noopener` to true.
9. Let `target browsing context` and `windowType` be the result of applying the rules for choosing a browsing context given `target`, `source browsing context`, and `noopener`.
10. Let `new` be true if `windowType` is either "new and unrestricted" or "new with no opener", and false otherwise.
11. If `target browsing context` is null, then return null.
12. If `new` is true, then set up browsing context features for `target browsing context` given `tokenizedFeatures`.
13. Let `urlRecord` be the URL `"about:blank"`.
14. If `url` is not the empty string or `new` is true, then:
   1. If `url` is not the empty string, then parse `url` relative to the `entry settings object`, and set `urlRecord` to the resulting URL record, if any. If the parse a URL algorithm failed, then throw a `SyntaxError` or `DOMException`.

---

Example

If there is a user agent that supports control-clicking a link to open it in a new tab, and the user control-clicks on an element whose `onclick` handler uses the `window.open()` API to open a page in an `iframe` element, the user agent could override the selection of the target browsing context to instead target a new tab.
2. Let request be a new request whose URL is urlRecord.

3. If noreferrer is true, then set request's referer to "noreferrer".

4. Let window be target browsing context's active window.

5. If urlRecord is "about:blank" and new is true, then queue a global task on the networking task source given window to fire an event named load at window, with the legacy target override flag set.

6. Otherwise:
   1. Let historyHandling be "replace" if new is true; otherwise "default".
   2. Navigate target browsing context to request, with exceptionsEnabled set to true, historyHandling set to historyHandling, and the source browsing context set to source browsing context.

15. Ifnoopener is true or windowType is "new with no opener", then return null.

16. Otherwise, if new is false, set target browsing context's opener browsing context to source browsing context.

   Note
   
   If new is true this is done as part of creating a new auxiliary browsing context.

17. Return target browsing context's WindowProxy object.

The open(url, target, features) 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 window open steps with url, target, and features.

To tokenize the features argument:

1. Let tokenizedFeatures be a new ordered map.

2. Let position point at the first code point of features.

3. While position is not past the end of features:
   1. Let name be the empty string.
   2. Let value be the empty string.
   3. Collect a sequence of code points that are feature separators from features given position. This skips past leading separators before the name.
   4. Collect a sequence of code points that are not feature separators from features given position. Set name to the collected characters, converted to ASCII lowercase.
   5. Set name to the result of normalizing the feature name name.
   6. While position is not past the end of features and the code point at position in features is not U+003D (=):
      1. If the code point at position in features is U+002C (,), or if it is not a feature separator, then break.
   2. Advance position by 1.

   Note
   
   This skips to the first U+003D (=) but does not skip past a U+002C (,) or a non-separator.

7. If the code point at position in features is a feature separator:
   1. While position is not past the end of features and the code point at position in features is a feature separator:
      1. If the code point at position in features is U+002C (,), then break.
   2. Advance position by 1.
2. **Collect a sequence of code points** that are not feature separators\(^{a831}\) code points from features given position. Set value to the collected code points, converted to ASCII lowercase.

8. If name is not the empty string, then set tokenizedFeatures[name] to value.

4. Return tokenizedFeatures.

A code point is a **feature separator** if it is ASCII whitespace, U+003D (=), or U+002C (,).

For legacy reasons, there are some aliases of some feature names. To **normalize a feature name** name, switch on name:

- ↪ "screenx"
  Return "left".
- ↪ "screeny"
  Return "top".
- ↪ "innerwidth"
  Return "width".
- ↪ "innerheight"
  Return "height".
- Anything else
  Return name.

To **parse a boolean feature** given a string value:

1. If value is the empty string, then return true.
2. If value is "yes", then return true.
3. Let parsed be the result of parsing value as an integer\(^{a68}\).
4. If parsed is an error, then set it to 0.
5. Return false if parsed is 0, and true otherwise.

The **name** attribute’s getter must run these steps:

1. If this Window\(^{a27}\) object’s browsing context\(^{a28}\) is null, then return the empty string.
2. Return this Window\(^{a27}\) object’s browsing context\(^{a28}\)’s name\(^{a22}\).

The **name**\(^{a31}\) attribute’s setter must run these steps:

1. If this Window\(^{a27}\) object’s browsing context\(^{a28}\) is null, then return.
2. Set this Window\(^{a27}\) object’s browsing context\(^{a28}\)’s name\(^{a22}\) to the given value.

**Note**

The name **gets reset**\(^{a890}\) when the browsing context is navigator\(^{a875}\) to another origin\(^{a840}\).

The **close()** method must run these steps:

1. Let current be this Window\(^{a27}\) object’s browsing context\(^{a28}\).
2. If current is null or its is closing\(^{a813}\) is true, then return.
3. If all the following are true
   - current is script-closable\(^{a832}\)
   - the incumbent global object\(^{a908}\)’s browsing context\(^{a28}\) is familiar with\(^{a820}\) current
the incumbent global object’s browsing context is allowed to navigate current

then:

1. Set current’s is closing to true.

2. Queue a task on the DOM manipulation task source to close current.

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’s getter must return true if this Window object’s browsing context is null or its is closing is true, and false otherwise.

The stop() method must stop document loading given this Window object’s associated Document.

The number of document-tree child browsing contexts of a Window object W is the result of running these steps:

1. If W’s browsing context is null, then return 0.

2. Return the number of document-tree child browsing contexts of W’s browsing context.

The length IDL attribute’s getter must return the number of document-tree child browsing contexts of this Window object.

Note

Indexed access to document-tree child browsing contexts is defined through the [[GetOwnProperty]] internal method of the WindowProxy object.

For web developers (non-normative)

window.length

Returns the number of document-tree child browsing contexts.

window[index]

Returns the indicated document-tree child browsing context.

7.3.3 Named access on the Window object

For web developers (non-normative)

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 document-tree child browsing context name property set of a Window object window is the return value of running these steps:

1. If window’s browsing context is null, then return the empty list.

2. Let childBrowsingContexts be all document-tree child browsing contexts of window’s browsing context whose browsing context name is not the empty string, in order, and including only the first document-tree child browsing context with a given name if multiple document-tree child browsing contexts have the same one.

3. Remove each browsing context from childBrowsingContexts whose active document’s origin is not same origin with window’s relevant settings object’s origin and whose browsing context name does not match the name of its container’s name content attribute value.
4. Return the **browsing context names**\(^{p822}\) of `childBrowsingContexts`, in the same order.

---

**Example**

This means that in the following example, hosted on https://example.org/, assuming https://elsewhere.example/ sets `window.name`\(^{p831}\) to "spices", evaluating `window.spices` after everything has loaded will yield undefined:

```html
<iframe src=https://elsewhere.example.com/></iframe>
<iframe name=spices></iframe>
```

---

The **Window**\(^{p827}\) object **supports named properties**. The **supported property names** of a **Window**\(^{p827}\) object `window` at any moment consist of the following, in **tree order** according to the element that contributed them, ignoring later duplicates:

- `window`'s **document-tree child browsing context name property set**\(^{p832}\);
- the value of the **name** content attribute for all **embed**\(^{p369}\), **form**\(^{p486}\), **img**\(^{p321}\), and **object**\(^{p373}\) elements that have a non-empty **name** content attribute and are **in a document tree** with `window`'s **associated Document**\(^{p828}\) as their **root**; and
- the value of the **id**\(^{p138}\) content attribute for all **HTML elements**\(^{p44}\) that have a non-empty **id**\(^{p138}\) content attribute and are **in a document tree** with `window`'s **associated Document**\(^{p828}\) as their **root**.

To **determine the value of a named property** `name` in a **Window**\(^{p827}\) object `window`, the user agent must return the value obtained using the following steps:

1. Let `objects` be the list of **named objects**\(^{p833}\) of `window` with the name `name`.

   **Note**

   *There will be at least one such object, by definition.*

2. If `objects` contains a **browsing context**\(^{p813}\), then return the **WindowProxy**\(^{p837}\) object of the first **browsing context container**\(^{p816}\) in **tree order** whose **nested browsing context**\(^{p816}\) is in `objects`.

3. Otherwise, if `objects` has only one element, return that element.

4. Otherwise return an **HTMLCollection** rooted at `window`'s **associated Document**\(^{p828}\) whose filter matches only **named objects**\(^{p833}\) of `window` with the name `name`. (By definition, these will all be elements.)

**Named objects** of a **Window**\(^{p827}\) object `window` with the name `name`, for the purposes of the above algorithm, consist of the following:

- **document-tree child browsing contexts**\(^{p816}\) of `window`'s **associated Document**\(^{p828}\) whose **name**\(^{p832}\) is `name`;
- **embed**\(^{p369}\), **form**\(^{p486}\), **img**\(^{p321}\), or **object**\(^{p373}\) elements that have a **name** content attribute whose value is `name` and are **in a document tree** with `window`'s **associated Document**\(^{p828}\) as their **root**; and
- **HTML elements**\(^{p44}\) that have an **id**\(^{p138}\) content attribute whose value is `name` and are **in a document tree** with `window`'s **associated Document**\(^{p828}\) as their **root**.

---

### 7.3.4 Discarding browsing contexts 5\(^{p83}\)

**To discard** a **Document**\(^{p715}\) `document`:

1. Set `document`'s **salvageable**\(^{p894}\) state to false.

2. Run any **unloading document cleanup steps**\(^{p895}\) for `document` that are defined by this specification and **other applicable specifications**\(^{p65}\).

3. **Abort**\(^{p896}\) `document`.

4. Remove any **tasks**\(^{p931}\) associated with `document` in any **task source**\(^{p932}\), without running those tasks.

5. **Discard**\(^{p834}\) all the **child browsing contexts**\(^{p816}\) of `document`.

6. For each **session history entry**\(^{p858}\) entry whose `document`\(^{p858}\) is equal to `document`, set entry's `document`\(^{p858}\) to null.

---

833
7. Set document's browsing context to null.

8. Remove document from the owner set of each WorkerGlobalScope object whose set contains document.

9. For each workletGlobalScope in document's worklet global scopes, terminate workletGlobalScope.

To discard a browsing context browsingContext, run these steps:

1. Discard all Document objects for all the entries in browsingContext's session history.

2. If browsingContext is a top-level browsing context, then remove browsingContext.

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, in addition to the other places where this specification requires them to be discarded.

7.3.5 Closing browsing contexts

To close a browsing context browsingContext, run these steps:

1. Prompt to unload browsingContext's active document. If the user refused to allow the document to be unloaded, then return.


3. Remove browsingContext from the user interface (e.g., close or hide its tab in a tabbed browser).


User agents should offer users the ability to arbitrarily close any top-level browsing context.

7.3.6 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:

```
[Exposed=Window]
interface BarProp {
  readonly attribute boolean visible;
};
```

For web developers (non-normative)

- `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 scrollbars are visible; otherwise, returns false.

- `window.statusbar.visible`
  Returns true if the status bar is visible; otherwise, returns false.
The `visible` attribute's getter must run these steps:

1. If this `BarProp` object's `relevant global object`'s `browsing context` is null, then return false.
2. If the user agent does not have a user interface element that the object represents, as described below, then return true.
3. Return 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 must exist for each `Window` object:

**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.

### 7.3.7 Script settings for `Window` objects

To set up a window environment settings object, given a URL `creationURL`, a JavaScript execution context, null or an environment `reservedEnvironment`, a URL `topLevelCreationURL`, and an origin `topLevelOrigin`, run these steps:
1. Let \(\text{realm}\) be the value of \(\text{execution context}'s \text{Realm component}\).

2. Let \(\text{window}\) be \(\text{realm}'s \text{global object}\).

3. Let \(\text{settings object}\) be a new \text{environment settings object} whose algorithms are defined as follows:

   \begin{itemize}
   \item The \textit{realm execution context}:
   \begin{itemize}
   \item Return \(\text{execution context}\).
   \end{itemize}
   \item The \textit{module map}:
   \begin{itemize}
   \item Return the \textit{module map} of \(\text{window}'s \text{associated Document}\).
   \end{itemize}
   \item The \textit{responsible document}:
   \begin{itemize}
   \item Return \(\text{window}'s \text{associated Document}\).
   \end{itemize}
   \item The \textit{API URL character encoding}:
   \begin{itemize}
   \item Return the \textit{current character encoding} of \(\text{window}'s \text{associated Document}\).
   \end{itemize}
   \item The \textit{API base URL}:
   \begin{itemize}
   \item Return the \textit{current base URL} of \(\text{window}'s \text{associated Document}\).
   \end{itemize}
   \item The \textit{origin}:
   \begin{itemize}
   \item Return the \textit{origin} of \(\text{window}'s \text{associated Document}\).
   \end{itemize}
   \item The \textit{referrer policy}:
   \begin{enumerate}
   \item Let \(\text{document}\) be \(\text{window}'s \text{associated Document}\).
   \item While \(\text{document}\) is \texttt{an iframe \texttt{srcdoc document}}, and \(\text{document}'s \text{referrer policy}\) is the empty string, set \(\text{document} to \text{document}'s \text{browsing context}''s \text{container document}.
   \item Return \(\text{document}'s \text{referrer policy}\).
   \end{enumerate}
   \item The \textit{embedder policy}:
   \begin{itemize}
   \item Return \(\text{window}'s \text{associated Document}''s \text{embedder policy}\).
   \end{itemize}
   \item The \textit{cross-origin isolated capability}:
   \begin{enumerate}
   \item \(\text{realm}'s \text{agent cluster}'s \text{cross-origin-isolation mode} \) is \texttt{"concrete"}, and
   \item \(\text{window}'s \text{associated Document}''s \text{target browsing context}''s \text{container document}''s \text{referrer policy} \) is \texttt{"cross-origin-isolated"}.
   \end{enumerate}
   \end{itemize}

4. If \(\text{reservedEnvironment}\) is non-null, then:

   \begin{enumerate}
   \item Set \(\text{settings object}'s \text{id}\) to \(\text{reservedEnvironment}'s \text{id}\), \(\text{target browsing context}\) to \(\text{reservedEnvironment}'s \text{target browsing context}\), and \(\text{active service worker}\) to \(\text{reservedEnvironment}'s \text{active service worker}\).
   \item Set \(\text{reservedEnvironment}'s \text{id}\) to the empty string.
   \end{enumerate}

\begin{note}
The identity of the reserved environment is considered to be fully transferred to the created \text{environment settings object}. The reserved environment is not searchable by the \text{environment}'s \text{id} from this point on.
\end{note}

5. Otherwise, set \(\text{settings object}'s \text{id}\) to a new unique opaque string, \(\text{settings object}'s \text{target browsing context}\) to null, and \(\text{settings object}'s \text{active service worker}\) to null.

6. Set \(\text{settings object}'s \text{creation URL}\) to \(\text{creationURL}\), \(\text{settings object}'s \text{top-level creation URL}\) to \(\text{topLevelCreationURL}\), and \(\text{settings object}'s \text{top-level origin}\) to \(\text{topLevelOrigin}\).

7. Set \(\text{realm}'s [[\text{HostDefined}]] \text{field to } \text{settings object}.
7.4 The WindowProxy exotic object

A WindowProxy is an exotic object that wraps a Window ordinary object, indiracting 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.

The WindowProxy exotic object must use the ordinary internal methods except where it is explicitly specified otherwise below.

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.

7.4.1 [[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.

7.4.2 [[SetPrototypeOf]] ( V )

1. Return ! SetImmutablePrototype(this, V).

7.4.3 [[IsExtensible]] ( )

1. Return true.

7.4.4 [[PreventExtensions]] ( )

1. Return false.

7.4.5 [[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 document-tree child browsing contexts of W.
   3. Let value be undefined.
   4. If maxProperties is greater than 0 and index is less than maxProperties, then set value to the WindowProxy object of the index document-tree child browsing context of W’s browsing context, sorted in the order that their browsing context container elements were most recently inserted into W’s associated Document, the WindowProxy object of the most recently inserted browsing context container’s nested browsing context.
being last.

5. If value is undefined, then:
   1. If \texttt{IsPlatformObjectSameOrigin}(W) is true, then return undefined.
   2. Throw a \texttt{"SecurityError" DOMException}.

6. Return \texttt{PropertyDescriptor} ( [Value]: value, [Writable]: false, [Enumerable]: true, [Configurable]: true ).

3. If \texttt{IsPlatformObjectSameOrigin}(W) is true, then return ! \texttt{OrdinaryGetOwnProperty}(W, P).

\begin{note}
This is a willful violation of the JavaScript specification's invariants of the essential internal methods to maintain compatibility with existing web content. See \texttt{tc39/ecma262 issue #672} for more information. \texttt{[JAVASCRIPT]}
\end{note}

4. Let property be ! \texttt{CrossOriginGetOwnPropertyHelper}(W, P).

5. If property is not undefined, then return property.

6. If property is undefined and P is in W's \texttt{document-tree child browsing context name property set}, then:
   1. Let value be the \texttt{WindowProxy} object of the \texttt{named object} of W with the name P.
   2. Return \texttt{PropertyDescriptor} ( [Value]: value, [Enumerable]: false, [Writable]: false, [Configurable]: true ).

\begin{note}
The reason the property descriptors are non-enumerable, despite this mismatching the same-origin behavior, is for compatibility with existing web content. See \texttt{issue #3183} for details.
\end{note}


\section*{7.4.6 [[DefineOwnProperty]] ( P, Desc )}

1. Let W be the value of the \texttt{[[Window]]} internal slot of this.

2. If \texttt{IsPlatformObjectSameOrigin}(W) is true, then:
   1. If P is an array index property name, return false.

\begin{note}
This is a willful violation of the JavaScript specification's invariants of the essential internal methods to maintain compatibility with existing web content. See \texttt{tc39/ecma262 issue #672} for more information. \texttt{[JAVASCRIPT]}
\end{note}

3. Throw a \texttt{"SecurityError" DOMException}.

\section*{7.4.7 [[Get]] ( P, Receiver )}

1. Let W be the value of the \texttt{[[Window]]} internal slot of this.

2. Check if an access between two browsing contexts should be reported, given the current global object's browsing context, W's browsing context, P, and the current settings object.

3. If \texttt{IsPlatformObjectSameOrigin}(W) is true, then return ? \texttt{OrdinaryGet}(this, P, Receiver).

7.4.8 [[Set]] (P, V, Receiver)  
1. Let W be the value of the [[Window]] internal slot of this.
2. Check if an access between two browsing contexts should be reported, given the current global object's browsing context, W's browsing context, P, and the current settings object.
3. If ! IsPlatformObjectSameOrigin(W) is true, then return ? OrdinarySet(this, P, V, Receiver).

Note this is passed rather than W as OrdinaryGet and CrossOriginGet will invoke the [[GetOwnProperty]] internal method.

7.4.9 [[Delete]] (P)  
1. Let W be the value of the [[Window]] internal slot of this.
2. If ! IsPlatformObjectSameOrigin(W) is true, then:
   1. If P is an array index property name, then:
      1. Let desc be ! this.[[GetOwnProperty]](P).
      2. If desc is undefined, then return true.
      3. Return false.
3. Throw a "SecurityError" DOMException.

7.4.10 [[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 document-tree 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).
7.5 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.

An origin is one of the following:

An opaque origin
An internal value, with no serialization it can be recreated from (it is serialized as "null" per serialization of an origin), 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.

The effective domain of an origin is computed as follows:

1. If origin is an opaque origin, then return null.
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 generally immutable. Only the domain of a tuple origin can be changed, and only through the document.domain API.

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 img 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.

Other specifications can override the above definitions by themselves specifying the origin of a particular image or media element.

The serialization of an origin is the string obtained by applying the following algorithm to the given 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.

Example

The serialization of ("https", "xn--maraa-rta.example", null, null) is "https://xn--maraa-rta.example".

Note

There used to also be a Unicode serialization of an origin. However, it was never widely adopted.

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 port 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

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>same origin</th>
<th>same origin-domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&quot;https&quot;, &quot;example.org&quot;, null, null)</td>
<td>(&quot;https&quot;, &quot;example.org&quot;, null, null)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(&quot;https&quot;, &quot;example.org&quot;, 314, null)</td>
<td>(&quot;https&quot;, &quot;example.org&quot;, 420, null)</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>(&quot;https&quot;, &quot;example.org&quot;, 314, &quot;example.org&quot;)</td>
<td>(&quot;https&quot;, &quot;example.org&quot;, 420, &quot;example.org&quot;)</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>(&quot;https&quot;, &quot;example.org&quot;, null, &quot;example.org&quot;)</td>
<td>(&quot;https&quot;, &quot;example.org&quot;, null, &quot;example.org&quot;)</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>(&quot;https&quot;, &quot;example.org&quot;, null, &quot;example.org&quot;)</td>
<td>(&quot;http&quot;, &quot;example.org&quot;, null, &quot;example.org&quot;)</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

7.5.1 Sites

A scheme-and-host is a tuple of a scheme and a host.

A site is an opaque origin or a scheme-and-host.

To obtain a site, given an origin origin, run these steps:

1. If origin is an opaque origin, then return origin.
2. If origin's host's registrable domain is null, then return (origin's scheme, origin's host).
3. Return (origin's scheme, origin's host's registrable domain).

Two origins, A and B, are said to be schemelessly same site 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, then:
   1. Let hostA be A's host, and let hostB be B's host.
2. If hostA equals hostB and hostA's registrable domain is null, then return true.

3. If hostA's registrable domain equals hostB's registrable domain and is non-null, then return true.

3. Return false.

Two origins, A and B, are said to be same site if both of the following statements are true:

- A and B are schemelessly same site.
- A and B are either both opaque origins, or both tuple origins with the same scheme.

Note: Unlike the same origin and same origin-domain concepts, for schemelessly same site and same site, the port and domain components are ignored.

⚠ Warning! For the reasons explained in URL, the same site and schemelessly same site concepts should be avoided when possible, in favor of same origin checks.

Example: Given that wildlife.museum, museum, and com are public suffixes and that example.com is not:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>schemelessly same site</th>
<th>same site</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&quot;https&quot;, &quot;example.com&quot;)</td>
<td>(&quot;https&quot;, &quot;sub.example.com&quot;)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(&quot;https&quot;, &quot;example.com&quot;)</td>
<td>(&quot;https&quot;, &quot;sub.other.example.com&quot;)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(&quot;https&quot;, &quot;example.com&quot;)</td>
<td>(&quot;http&quot;, &quot;non-secure.example.com&quot;)</td>
<td>✓</td>
<td>❌</td>
</tr>
<tr>
<td>(&quot;https&quot;, &quot;r.wildlife.museum&quot;)</td>
<td>(&quot;https&quot;, &quot;sub.r.wildlife.museum&quot;)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(&quot;https&quot;, &quot;r.wildlife.museum&quot;)</td>
<td>(&quot;https&quot;, &quot;sub.other.r.wildlife.museum&quot;)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(&quot;https&quot;, &quot;r.wildlife.museum&quot;)</td>
<td>(&quot;https&quot;, &quot;other.wildlife.museum&quot;)</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>(&quot;https&quot;, &quot;r.wildlife.museum&quot;)</td>
<td>(&quot;https&quot;, &quot;wildlife.museum&quot;)</td>
<td>✓</td>
<td>❌</td>
</tr>
<tr>
<td>(&quot;https&quot;, &quot;wildlife.museum&quot;)</td>
<td>(&quot;https&quot;, &quot;wildlife.museum&quot;)</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

(Here we have omitted the port and domain components since they are not considered.)

7.5.2 Relaxing the same-origin restriction

For web developers (non-normative)

**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. This enables pages on different hosts of a domain to synchronously access each other's DOMs.

In sandboxed iframes, Document's with opaque origins, Document's without a browsing context, and when the "document-domain" feature is disabled, the setter will throw a "SecurityError" exception. In cases where crossOriginIsolated or originAgentCluster return true, the setter will do nothing.

Avoid using the document.domain setter. It undermines the security protections provided by the same-origin policy. This is especially acute when using shared hosting; for example, 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 setter has been used.

Because of these security pitfalls, this feature is in the process of being removed from the web platform. (This is a long process that takes many years.)
Instead, use `postMessage()` or `MessageChannel` objects to communicate across origins in a safe manner.

The `domain` getter steps are:

1. Let `effectiveDomain` be this's origin's effective domain.
2. If `effectiveDomain` is null, then return the empty string.
3. Return `effectiveDomain`, serialized.

The `domain` setter steps are:

1. If this's browsing context is null, then throw a "SecurityError" DOMException.
2. If this's active sandboxing flag set has its sandboxed document.domain browsing context flag set, then throw a "SecurityError" DOMException.
3. If this is not allowed to use the "document-domain" feature, then throw a "SecurityError" DOMException.
4. Let `effectiveDomain` be this's origin's effective domain.
5. If `effectiveDomain` is null, then throw a "SecurityError" DOMException.
6. If the given value is not a registrable domain suffix of and is not equal to `effectiveDomain`, then throw a "SecurityError" DOMException.
7. If the surrounding agent's agent cluster's is origin-keyed is true, then return.
8. Set this's origin's domain to the result of parsing the given value.

To determine if a string `hostSuffixString` is a registrable domain suffix of or is equal to a `host originalHost`, run these steps:

1. If `hostSuffixString` is the empty string, then return false.
2. Let `host` be the result of parsing `hostSuffixString`.
3. If `host` is failure, then return false.
4. If `host` does not equal `originalHost`, then:
   1. If `host` or `originalHost` is not a domain, then return false.
   2. If `host`, prefixed by a U+002E FULL STOP (.), does not exactly match the end of `originalHost`, then return false.
   3. If `host` equals `host`'s public suffix, then return false. [URL]
5. Return true.

### 7.5.3 Origin-keyed agent clusters

For web developers (non-normative)

`window.originAgentCluster()`

Returns true if this `Window` belongs to an agent cluster which is origin-keyed, in the manner described in this section.

A `Document` delivered over a secure context can request that it be placed in an origin-keyed agent cluster, by using the `Origin-Agent-Cluster` HTTP response header. This header is a structured header whose value must be a boolean. [STRUCTURED-FIELDS]

Per the processing model in the create and initialize a new `Document` object, values that are not the structured header boolean true
value (i.e., `?1`) will be ignored.

The consequences of using this header are that the resulting Document's agent cluster key is its origin, instead of the corresponding site. In terms of observable effects, this means that attempting to relax the same-origin restriction using document.domain will instead do nothing, and it will not be possible to send WebAssembly.Module objects to cross-origin Document objects (even if they are same site). Behind the scenes, this isolation can allow user agents to allocate implementation-specific resources corresponding to agent clusters, such as processes or threads, more efficiently.

Note that within a browsing context group, the `Origin-Agent-Cluster` header can never cause same-origin Document objects to end up in different agent clusters, even if one sends the header and the other doesn’t. This is prevented by means of the historical agent cluster key map.

The `originAgentCluster` getter steps are to return the surrounding agent's agent cluster's is origin-keyed.

Note

This means that the `originAgentCluster` getter can return false, even if the header is set, if the header was omitted on a previously-loaded same-origin page in the same browsing context group. Similarly, it can return true even when the header is not set.

The `originAgentCluster` getter steps are to return the surrounding agent’s agent cluster’s is origin-keyed.

Note

Document's with an opaque origin can be considered unconditionally origin-keyed; for them the header has no effect, and the `originAgentCluster` getter will always return true.

Note

Similarly, Document's whose agent cluster's cross-origin isolation mode is not "none" are automatically origin-keyed. The `Origin-Agent-Cluster` header might be useful as an additional hint to implementations about resource allocation, since the `Cross-Origin-Opener-Policy` and `Cross-Origin-Embedder-Policy` headers used to achieve cross-origin isolation are more about ensuring that everything in the same address space opts in to being there. But adding it would have no additional observable effects on author code.

7.6 Sandboxning

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 without user activation browsing context flag and sandboxed top-level navigation with user activation 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 or the `window.open()` method.

The sandboxed top-level navigation without user activation browsing context flag

This flag prevents content from navigating their top-level browsing context and prevents content from closing their top-level browsing context. It is consulted only when the sandboxed browsing context's active window does not have transient activation.

When the sandboxed top-level navigation without user activation browsing context flag is not set, content can navigate its top-
The sandboxed top-level navigation with user activation browsing context flag
This flag prevents content from navigating their top-level browsing context and prevents content from closing their top-level browsing context. It is consulted only when the sandboxed browsing context's active window has transient activation.

As with the sandboxed top-level navigation without user activation browsing context flag, this flag only affects the top-level browsing context; if it is not set, other browsing contexts might still be protected by other flags.

The sandboxed plugins browsing context flag
This flag prevents content from instantiating plugins, whether using the embed element, the object element, or through navigation of their 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.

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.

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 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()` [964]
- `window.confirm()` [964]
- `window.print()` [965]
- `window.prompt()` [965]
- the `beforeunload` event

The sandboxed orientation lock browsing context flag
This flag disables the ability to lock the screen orientation. [SCREENORIENTATION]

The sandboxed presentation browsing context flag
This flag disables the Presentation API. [PRESENTATION]

The sandboxed downloads browsing context flag
This flag prevents content from initiating or instantiating downloads, whether through downloading hyperlinks or through navigation that gets handled as a download.

When the user agent is to parse a sandboxing directive, given a string input, a sandboxing flag set output, it must run the
following steps:

1. **Split input on ASCII whitespace**, to obtain tokens.
2. Let output be empty.
3. Add the following flags to output:
   - The sandboxed navigation browsing context flag\(^{844}\).
   - The sandboxed auxiliary navigation browsing context flag\(^{844}\), unless tokens contains the allow-popups keyword.
   - The sandboxed top-level navigation without user activation browsing context flag\(^{844}\), unless tokens contains the allow-top-navigation keyword.
   - The sandboxed top-level navigation with user activation browsing context flag\(^{844}\), unless tokens contains either the allow-top-navigation-by-user-activation keyword or the allow-top-navigation\(^{846}\) keyword.

   **Note**
   This means that if the allow-top-navigation\(^{846}\) is present, the allow-top-navigation-by-user-activation\(^{846}\) keyword will have no effect. For this reason, specifying both is a document conformance error.

   - The sandboxed plugins browsing context flag\(^{845}\).
   - The sandboxed origin browsing context flag\(^{845}\), unless the tokens contains the allow-same-origin keyword.

   **Note**
   The allow-same-origin\(^{846}\) 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 popups, 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\(^{845}\), unless tokens contains the allow-forms keyword.
   - The sandboxed pointer lock browsing context flag\(^{845}\), unless tokens contains the allow-pointer-lock keyword.
   - The sandboxed scripts browsing context flag\(^{845}\), unless tokens contains the allow-scripts keyword.
   - The sandboxed automatic features browsing context flag\(^{845}\), unless tokens contains the allow-scripts\(^{846}\) 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 document.domain browsing context flag\(^{845}\).
   - The sandbox propagates to auxiliary browsing contexts flag\(^{845}\), unless tokens contains the allow-popups-to-escape-sandbox keyword.
   - The sandboxed modals flag\(^{845}\), unless tokens contains the allow-modals keyword.
   - The sandboxed orientation lock browsing context flag\(^{845}\), unless tokens contains the allow-orientation-lock keyword.
   - The sandboxed presentation browsing context flag\(^{845}\), unless tokens contains the allow-presentation keyword.
   - The sandboxed downloads browsing context flag\(^{845}\), unless tokens contains the allow-downloads keyword.

Every top-level browsing context\(^{816}\) has a **pop-up sandboxing flag set**, which is a sandboxing flag set\(^{844}\). When a browsing context\(^{813}\) is created, its popup sandboxing flag set\(^{846}\) must be empty. It is populated by the rules for choosing a browsing context\(^{822}\).
and the obtain a browsing context to use for a navigation response\textsuperscript{850} algorithm.

Every \texttt{iframe}\textsuperscript{362} element has an \textbf{iframe sandboxing flag set}, which is a sandboxing flag set\textsuperscript{844}. Which flags in an \texttt{iframe} sandboxing flag set\textsuperscript{847} are set at any particular time is determined by the \texttt{iframe}\textsuperscript{362} element's sandbox\textsuperscript{856} attribute.

Every \texttt{Document}\textsuperscript{115} has an \textbf{active sandboxing flag set}, which is a sandboxing flag set\textsuperscript{844}. When the \texttt{Document}\textsuperscript{115} is created, its active sandboxing flag set\textsuperscript{847} must be empty. It is populated by the navigation algorithm\textsuperscript{875}.

Every resource that is obtained by the navigation algorithm\textsuperscript{875} has a \textbf{forced sandboxing flag set}, which is a sandboxing flag set\textsuperscript{844}. A resource by default has no flags set in its forced sandboxing flag set\textsuperscript{847}, but other specifications can define that certain flags are set.

To determine the creation sandboxing flags for a browsing context\textsuperscript{613} browsing context, given null or an element embedder, return the \texttt{union} of the flags that are present in the following sandboxing flag sets\textsuperscript{844}:

- If embedder is null, then: the flags set on browsing context's popup sandboxing flag set\textsuperscript{846}.
- If embedder is an element, then: the flags set on embedder's iframe sandboxing flag set\textsuperscript{847}.
- If embedder is an element, then: the flags set on embedder's node document's active sandboxing flag set\textsuperscript{847}.

After creation, the sandboxing flags for a browsing context\textsuperscript{613} browsing context are the result of determining the creation sandboxing flags\textsuperscript{847} given browsing context and browsing context's container\textsuperscript{816}.

### 7.7 Cross-origin opener policies

A \textbf{cross-origin opener policy value} allows a document which is navigated to in a top-level browsing context\textsuperscript{816} to force the creation of a new top-level browsing context\textsuperscript{816}, and a corresponding group\textsuperscript{817}. The possible values are:

- **"unsafe-none"**
  - This is the (current) default and means that the document will occupy the same top-level browsing context\textsuperscript{816} as its predecessor, unless that document specified a different cross-origin opener policy\textsuperscript{847}.

- **"same-origin-allow-popups"**
  - This forces the creation of a new top-level browsing context\textsuperscript{816} for the document, unless its predecessor specified the same cross-origin opener policy\textsuperscript{847} and they are same origin\textsuperscript{841}.

- **"same-origin"**
  - This behaves the same as "same-origin-allow-popups\textsuperscript{847}", with the addition that any auxiliary browsing context\textsuperscript{827} created needs to contain same origin\textsuperscript{841} documents that also have the same cross-origin opener policy\textsuperscript{847} or it will appear closed to the opener.

- **"same-origin-plus-COEP"**
  - This behaves the same as "same-origin\textsuperscript{841}", with the addition that it sets the (new) top-level browsing context's group\textsuperscript{817}’s cross-origin isolation mode\textsuperscript{821} to one of "logical\textsuperscript{821}" or "concrete\textsuperscript{821}".

\textbf{Note}

"same-origin-plus-COEP\textsuperscript{847}" cannot be directly set via the `Cross-Origin-Opener-Policy\textsuperscript{1242}` header, but results from a combination of setting both `Cross-Origin-Opener-Policy\textsuperscript{1242}`: same-origin\textsuperscript{847}, and `Cross-Origin-Embedder-Policy\textsuperscript{1242}`: require-corp together.

A \textbf{cross-origin opener policy} consists of:

- A \textit{value}, which is a cross-origin opener policy value\textsuperscript{847}, initially "unsafe-none\textsuperscript{847}".
- A \textit{reporting endpoint}, which is string or null, initially null.
• A **report-only value**, which is a cross-origin opener policy value, initially "unsafe-none".

• A **report-only reporting endpoint**, which is a string or null, initially null.

To **match cross-origin opener policy values**, given a cross-origin opener policy value A, an origin originA, a cross-origin opener policy value B, and an origin originB:

1. If A is "unsafe-none" and B is "unsafe-none", then return true.
2. If A is "unsafe-none" or B is "unsafe-none", then return false.
3. If A is B and originA is **same origin** with originB, then return true.
4. Return false.

### 7.7.1 The headers

A **Document**'s cross-origin opener policy is derived from the `Cross-Origin-Opener-Policy` and the `Cross-Origin-Opener-Policy-Report-Only` HTTP response headers. These headers are **structured headers** whose value must be a **token**.

The valid **token** values are the opener policy values. The token may also have attached parameters; of these, the "report-to" parameter can have a valid URL string identifying an appropriate reporting endpoint.

**Note**

Per the processing model described below, user agents will ignore this header if it contains an invalid value. Likewise, user agents will ignore this header if the value cannot be parsed as a **token**.

To **obtain a cross-origin opener policy** given a response response and an environment reservedEnvironment:

1. Let policy be a new cross-origin opener policy.
2. If reservedEnvironment is a non-secure context, then return policy.
3. Let value be the result of getting a structured field value given `Cross-Origin-Opener-Policy` and "item" from response's header list.
4. If parsedItem is not null, then:
   1. If parsedItem[0] is "same-origin", then:
      1. Let coep be the result of obtaining a cross-origin embedder policy from response and reservedEnvironment.
      2. If coep's value is "require-corp", then set policy's value to "same-origin-plus-COEP".
      3. Otherwise, set policy's value to "same-origin".
   2. If parsedItem[0] is "same-origin-allow-popups", then set policy's value to "same-origin-allow-popups".
   3. If parsedItem[1]["report-to"] exists and it is a string, then set policy's reporting endpoint to parsedItem[1]["report-to"].
5. Set parsedItem to the result of getting a structured field value given `Cross-Origin-Opener-Policy-Report-Only` and "item" from response's header list.
6. If parsedItem is not null, then:
   1. If parsedItem[0] is "same-origin", then:
      1. Let coep be the result of obtaining a cross-origin embedder policy from response and reservedEnvironment.
2. If coep's value is "require-corp" or coep's report-only value is "require-corp", then set policy's report-only value to "same-origin-plus-COEP".

Note
Report only COOP also considers report-only COEP to assign the special "same-origin-plus-COEP" value. This allows developers more freedom in the order of deployment of COOP and COEP.

3. Otherwise, set policy's report-only value to "same-origin".

2. If parsedItem[0] is "same-origin-allow-popups", then set policy's report-only value to "same-origin-allow-popups".

3. If parsedItem[1]"report-to"] exists and it is a string, then set policy's report-only reporting endpoint to parsedItem[1]"report-to"].

7. Return policy.

7.7.2 Browsing context group switches due to cross-origin opener policy

To check if COOP values require a browsing context group switch, given a boolean isInitialAboutBlank, two origins responseOrigin, activeDocumentNavigationOrigin, and two cross-origin opener policy values responseCOOPValue and activeDocumentCOOPValue:

1. If the result of matching activeDocumentCOOPValue, activeDocumentNavigationOrigin, responseCOOPValue, and responseOrigin is true, return false.

2. If all of the following are true:
   ◦ isInitialAboutBlank,
   ◦ activeDocumentCOOPValue's value is "same-origin-allow-popups",
   ◦ responseCOOPValue is "unsafe-none",

   then return false.

3. Return true.

To check if enforcing report-only COOP would require a browsing context group switch, given a boolean isInitialAboutBlank, two origins responseOrigin, activeDocumentNavigationOrigin, and two cross-origin opener policies responseCOOP and activeDocumentCOOP:

1. If the result of checking if COOP values require a browsing context group switch, given isInitialAboutBlank, responseOrigin, activeDocumentNavigationOrigin, responseCOOP's report-only value and activeDocumentCOOPReportOnly's report-only value is false, then return false.

Note
Matching report-only policies allows a website to specify the same report-only cross-origin opener policy on all its pages and not receive violation reports for navigations between these pages.

2. If the result of checking if COOP values require a browsing context group switch, given isInitialAboutBlank, responseOrigin, activeDocumentNavigationOrigin, responseCOOP's value and activeDocumentCOOPReportOnly's report-only value is true, then return true.

3. If the result of checking if COOP values require a browsing context group switch, given isInitialAboutBlank, responseOrigin, activeDocumentNavigationOrigin, responseCOOP's report-only value and activeDocumentCOOPReportOnly's value is true, then return true.

4. Return false.

A cross-origin opener policy enforcement result is a struct with the following items:

- A boolean needs a browsing context group switch, initially false.
- A boolean would need a browsing context group switch due to report-only, initially false.
- A URL url.
• An origin current origin.
• A cross-origin opener policy cross-origin opener policy.
• A boolean current context is navigation source.

To enforce a response's cross-origin opener policy, given a browsing context, a URL responseURL, an origin responseOrigin, a cross-origin opener policy responseCOOP, a cross-origin opener policy enforcement result currentCOOPEnforcementResult, and a referer referer:

1. Let newCOOPEnforcementResult be a new cross-origin opener policy enforcement result whose needs a browsing context group switch is currentCOOPEnforcementResult's needs a browsing context group switch would need a browsing context group switch due to report-only is currentCOOPEnforcementResult's would need a browsing context group switch due to report-only url is responseURL, origin is responseOrigin, coop is responseCOOP, and current context is navigation source is true.

2. Let isInitialAboutBlank be false.

3. If browsingContext's only entry in its session history is the about:blank Document that was added when browsingContext was created, then set isInitialAboutBlank to true.

4. If isInitialAboutBlank is true and browsingContext's initial URL is null, set browsingContext's initial URL to responseURL.

5. If the result of checking if COOP values require a browsing context group switch given isInitialAboutBlank, currentCOOPEnforcementResult's cross-origin opener policy's value, currentCOOPEnforcementResult's origin, responseCOOP's value, and responseOrigin is true, then:
   1. Set newCOOPEnforcementResult's needs a browsing context group switch to true.
   2. If browsingContext's group's browsing context set's size is greater than 1, then:
      1. Queue a violation report for browsing context group switch when navigating to a COOP response with responseCOOP, "enforce", responseURL, currentCOOPEnforcementResult's url, currentCOOPEnforcementResult's origin, responseOrigin, and referer.
      2. Queue a violation report for browsing context group switch when navigating away from a COOP response with activeDocumentCOOP, "enforce", currentCOOPEnforcementResult's url, responseURL, currentCOOPEnforcementResult's origin, responseOrigin, and currentCOOPEnforcementResult's current context is navigation source.

6. If the result of checking if enforcing report-only COOP would require a browsing context group switch given isInitialAboutBlank, responseOrigin, currentCOOPEnforcementResult's origin, responseCOOP, and currentCOOPEnforcementResult's cross-origin opener policy is true, then:
   1. Set result's would need a browsing context group switch due to report-only to true.
   2. If browsingContext's group's browsing context set's size is greater than 1, then:
      1. Queue a violation report for browsing context group switch when navigating to a COOP response with responseCOOP, "reporting", responseURL, currentCOOPEnforcementResult's url, currentCOOPEnforcementResult's origin, responseOrigin, and referer.
      2. Queue a violation report for browsing context group switch when navigating away from a COOP response with activeDocumentCOOP, "reporting", currentCOOPEnforcementResult's url, responseURL, currentCOOPEnforcementResult's origin, responseOrigin, and currentCOOPEnforcementResult's current context is navigation source.

7. Return newCOOPEnforcementResult.

To obtain a browsing context to use for a navigation response, given a browsing contexts, a sandboxing flag set sandboxFlags, a cross-origin opener policy, navigationCOOP, and a cross-origin opener policy enforcement result coopEnforcementResult:

1. Assert: browsingContext is a top-level browsing context.
2. If coopEnforcementResult's needs a browsing context group switch is false, then:
1. If coopEnforcementResult's would need a browsing context group switch due to report-only is true, set browsing context's virtual browsing context group ID to a new unique identifier.

2. Return browsingContext.

3. Let newBrowsingContext be the result of creating a new top-level browsing context.

4. If navigationCOOP's value is "same-origin-plus-COEP", then set newBrowsingContext's cross-origin isolation mode to either "logical" or "concrete". The choice of which is implementation-defined.

   Note
   It is difficult on some platforms to provide the security properties required by the cross-origin isolated capability. "concrete" grants access to it and "logical" does not.

5. If sandboxFlags is not empty, then:
   1. Assert navigationCOOP's value is "unsafe-none".
   2. Assert: newBrowsingContext's popup sandboxing flag set is empty.
   3. Set newBrowsingContext's popup sandboxing flag set to a clone of sandboxFlags.


   Note
   This has no effect on browsingContext's group, unless browsingContext was its sole top-level browsing context. In that case, the user agent might delete the browsing context group which no longer contains any browsing contexts.

7. Return newBrowsingContext.

The impact of swapping browsing context groups following a navigation is not fully defined. It is currently under discussion in issue #5350.

7.7.3 Reporting

An accessor-accessed relationship is an enum that describes the relationship between two browsing contexts between which an access happened. It can take the following values:

accessor is opener
The accessor or one of its ancestors is the opener browsing context of the accessed browsing context's top-level browsing context.

accessor is openee
The accessed or one of its ancestors is the opener browsing context of the accessor browsing context's top-level browsing context.

none
There is no opener relationship between the accessor, the accessor, or any of their ancestors.

To check if an access between two browsing contexts should be reported, given two browsing contexts accessor and accessed, a JavaScript property name P, and an environment settings object:

1. If P is not a cross-origin accessible window property name, then return.

2. If accessor's active document's origin or any of its ancestors, active document's origins are not same origin, with accessor's top-level browsing context's active document's origin, or if accessed's active document's origin or any of its ancestors, active document's origins are not same origin, with accessed's top-level browsing context's active document's origin, then return.
3. If accessor's top-level browsing context's virtual browsing context group ID is accessed's top-level browsing context's virtual browsing context group ID, then return.


5. If accessed's top-level browsing context's opener browsing context is accessor or an ancestor of accessor, then set accessorAccessedRelationship to accessor is opener.

6. If accessor's top-level browsing context's opener browsing context is accessed or an ancestor of accessed, then set accessorAccessedRelationship to accessor is openee.


To sanitize a URL to send in a report given a URL url:

1. Let sanitizedURL be a copy of url.
2. Set the username given sanitizedURL and the empty string.
3. Set the password given sanitizedURL and the empty string.
4. Return the serialization of sanitizedURL with exclude fragment set to true.

To queue a violation report for browsing context group switch when navigating to a COOP response given a cross-origin opener policy, coop, a string disposition, a URL coopURL, a URL previousResponseURL, two origins coopOrigin and previousResponseOrigin, and a referrer referrer:

1. If coop's reporting endpoint is null, return.
2. Let coopValue be coop's value.
3. If disposition is "reporting", then set coopValue to coop's report-only value.
4. Let serializedReferrer be an empty string.
5. If referrer is a URL, set serializedReferrer to the serialization of referrer.
6. Let body be a new object containing the following properties:

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>disposition</td>
<td>disposition</td>
</tr>
<tr>
<td>effectivePolicy</td>
<td>coopValue</td>
</tr>
<tr>
<td>previousResponseURL</td>
<td>If coopOrigin and previousResponseOrigin are same origin then this is the sanitation of previousResponseURL, null otherwise. Otherwise, serializedReferrer</td>
</tr>
<tr>
<td>referrer</td>
<td>serializedReferrer</td>
</tr>
<tr>
<td>type</td>
<td>&quot;navigation-to-response&quot;</td>
</tr>
</tbody>
</table>

7. Queue body as "coop" for coop's reporting endpoint with coopURL.

To queue a violation report for browsing context group switch when navigating away from a COOP response given a cross-origin opener policy, coop, a string disposition, a URL coopURL, a URL nextResponseURL, two origins coopOrigin and nextResponseOrigin, and a boolean isCOOPResponseNavigationSource:

1. If coop's reporting endpoint is null, return.
2. Let coopValue be coop's value.
3. If disposition is "reporting", then set coopValue to coop's report-only value.

4. Let body be a new object containing the following properties:

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>disposition</td>
<td>disposition</td>
</tr>
<tr>
<td>effectivePolicy</td>
<td>coopValue</td>
</tr>
<tr>
<td>nextResponseURL</td>
<td>If coopOrigin and nextResponseOrigin are same origin, or isCOOPResponseNavigationSource is true, this is the sanitization of previousResponseURL, null otherwise.</td>
</tr>
<tr>
<td>type</td>
<td>&quot;navigation-to-response&quot;</td>
</tr>
</tbody>
</table>

5. Queue body as "coop" for coop's reporting endpoint with coopURL.

To queue violation reports for accesses, given an accessor-accessed relationship, two cross-origin opener policies, accessorCOOP and accessedCOOP, four URLs accessorURL, accessedURL, accessorInitialURL, accessedInitialURL, four origins accessorOrigin, accessedOrigin, accessorCreatorOrigin and accessedCreatorOrigin, two referrers accessorReferrer and accessedReferrer, a string propertyName, and an environment settings object environment:

1. If coop's reporting endpoint is null, return.

2. Let coopValue be coop's value.

3. If disposition is "reporting", then set coopValue to coop's report-only value.

4. If accessorAccessedRelationship is accessor is opener:

   1. Queue a violation report for access to an opened window, given accessorCOOP, accessorURL, accessedURL, accessorInitialURL, accessorOrigin, accessedOrigin, accessorCreatorOrigin, propertyName, and environment.


5. Otherwise, if accessorAccessedRelationship is accessor is openee:

   1. Queue a violation report for access to the opener, given accessorCOOP, accessorURL, accessedURL, accessorOrigin, accessedOrigin, propertyName, accessorReferrer, and environment.

   2. Queue a violation report for access from an opened window, given accessedCOOP, accessedURL, accessorURL, accessorInitialURL, accessorOrigin, accessorOrigin, accessorCreatorOrigin, and propertyName.

6. Otherwise:

   1. Queue a violation report for access to another window, given accessorCOOP, accessorURL, accessedURL, accessorOrigin, accessedOrigin, propertyName, and environment.

   2. Queue a violation report for access from another window, given accessedCOOP, accessedURL, accessorURL, accessedOrigin, accessorOrigin, and propertyName.

To queue a violation report for access to the opener, given a cross-origin opener policy coop, two URLs coopURL and openerURL, two origins coopOrigin and openerOrigin, a string propertyName, a referrer referrer, and an environment settings object environment:

1. Let sourceFile, lineNumber and columnNumber be the relevant script URL and problematic position which triggered this report.

2. Let serializedReferrer be an empty string

3. If referrer is a URL, set serializedReferrer to the serialization of referrer.

4. Let body be a new object containing the following properties:

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>disposition</td>
<td>&quot;reporting&quot;</td>
</tr>
<tr>
<td>effectivePolicy</td>
<td>coop's report-only value</td>
</tr>
<tr>
<td>property</td>
<td>propertyName</td>
</tr>
<tr>
<td>openerURL</td>
<td>If coopOrigin and openerOrigin are same origin, this is the sanitization of openerURL, null otherwise.</td>
</tr>
<tr>
<td>referrer</td>
<td>serializedReferrer</td>
</tr>
</tbody>
</table>
To queue a violation report for access to an opened window, given a cross-origin opener policy coop, three URLs coopURL, openedWindowURL and initialWindowURL, three origins coopOrigin, openedWindowOrigin, and openerInitialOrigin, a string propertyName, and an environment settings object, environment:

1. Let sourceFile, lineNumber and columnNumber be the relevant script URL and problematic position which triggered this report.

2. Let body be a new object containing the following properties:

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>disposition</td>
<td>&quot;reporting&quot;</td>
</tr>
<tr>
<td>effectivePolicy</td>
<td>coop's report-only value</td>
</tr>
<tr>
<td>property</td>
<td>propertyName</td>
</tr>
<tr>
<td>openedWindowURL</td>
<td>If coopOrigin and openedWindowOrigin are same origin, this is the sanitization of openedWindowURL, null otherwise.</td>
</tr>
<tr>
<td>openedWindowInitialURL</td>
<td>If coopOrigin and openerInitialOrigin are same origin, this is the sanitization of initialWindowURL, null otherwise.</td>
</tr>
<tr>
<td>sourceFile</td>
<td>sourceFile</td>
</tr>
<tr>
<td>lineNumber</td>
<td>lineNumber</td>
</tr>
<tr>
<td>columnNumber</td>
<td>columnNumber</td>
</tr>
<tr>
<td>type</td>
<td>&quot;access-to-opener&quot;</td>
</tr>
</tbody>
</table>

3. Queue body as "coop" for coop's reporting endpoint with coopURL and environment.

To queue a violation report for access to another window, given a cross-origin opener policy coop, two URLs coopURL and otherURL, two origins coopOrigin and otherOrigin, a string propertyName, and an environment settings object, environment:

1. Let sourceFile, lineNumber and columnNumber be the relevant script URL and problematic position which triggered this report.

2. Let body be a new object containing the following properties:

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>disposition</td>
<td>&quot;reporting&quot;</td>
</tr>
<tr>
<td>effectivePolicy</td>
<td>coop's report-only value</td>
</tr>
<tr>
<td>property</td>
<td>propertyName</td>
</tr>
<tr>
<td>otherURL</td>
<td>If coopOrigin and otherOrigin are same origin, this is the sanitization of otherURL, null otherwise.</td>
</tr>
<tr>
<td>sourceFile</td>
<td>sourceFile</td>
</tr>
<tr>
<td>lineNumber</td>
<td>lineNumber</td>
</tr>
<tr>
<td>columnNumber</td>
<td>columnNumber</td>
</tr>
<tr>
<td>type</td>
<td>&quot;access-to-opener&quot;</td>
</tr>
</tbody>
</table>

3. Queue body as "coop" for coop's reporting endpoint with coopURL and environment.

To queue a violation report for access from the opener, given a cross-origin opener policy coop, two URLs coopURL and openerURL, two origins coopOrigin and openerOrigin, a string propertyName, and a referrer:

1. If coop's reporting endpoint is null, return.

2. Let serializedReferrer be an empty string

3. If referrer is a URL, set serializedReferrer to the serialization of referrer.

4. Let body be a new object containing the following properties:

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>disposition</td>
<td>&quot;reporting&quot;</td>
</tr>
</tbody>
</table>
5. Queue body as "coop" for coop's reporting endpoint\(^\text{847}\) with coopURL.

To queue a violation report for access from an opened window, given a cross-origin opener policy\(^\text{847}\) coop, three URLs coopURL, openedWindowURL and initialWindowURL, three origins\(^\text{840}\) coopOrigin, openedWindowOrigin, and openerInitialOrigin, and a string propertyName:

1. If coop's reporting endpoint\(^\text{847}\) is null, return.
2. Let body be a new object containing the following properties:

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>disposition</td>
<td>&quot;reporting&quot;</td>
</tr>
<tr>
<td>effectivePolicy</td>
<td>coopValue</td>
</tr>
<tr>
<td>property</td>
<td>propertyName</td>
</tr>
<tr>
<td>openedWindowURL</td>
<td>If coopOrigin and openedWindowOrigin are same origin(^\text{844}), this is the sanitization(^\text{852}) of openedWindowURL, null otherwise.</td>
</tr>
<tr>
<td>openedWindowInitialURL</td>
<td>If coopOrigin and openerInitialOrigin are same origin(^\text{844}), this is the sanitization(^\text{852}) of initialWindowURL, null otherwise.</td>
</tr>
<tr>
<td>type</td>
<td>&quot;access-to-opener&quot;</td>
</tr>
</tbody>
</table>

3. Queue body as "coop" for coop's reporting endpoint\(^\text{847}\) with coopURL.

To queue a violation report for access from another window, given a cross-origin opener policy\(^\text{847}\) coop, two URLs coopURL and otherURL, two origins\(^\text{840}\) coopOrigin and otherOrigin, and a string propertyName:

1. If coop's reporting endpoint\(^\text{847}\) is null, return.
2. Let body be a new object containing the following properties:

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>disposition</td>
<td>&quot;reporting&quot;</td>
</tr>
<tr>
<td>effectivePolicy</td>
<td>coop's report-only value(^\text{842})</td>
</tr>
<tr>
<td>property</td>
<td>propertyName</td>
</tr>
<tr>
<td>otherURL</td>
<td>If coopOrigin and otherOrigin are same origin(^\text{844}), this is the sanitization(^\text{852}) of otherURL, null otherwise.</td>
</tr>
<tr>
<td>type</td>
<td>access-to-opener</td>
</tr>
</tbody>
</table>

3. Queue body as "coop" for coop's reporting endpoint\(^\text{847}\) with coopURL.

7.8 Cross-origin embedder policies \(^\text{855}\)

An embedder policy value controls the fetching of cross-origin resources without explicit permission from resource owners. There are two such values:

"unsafe-none"

This is the default value. When this value is used, cross-origin resources can be fetched without giving explicit permission through the CORS protocol or the `Cross-Origin-Resource-Policy` header.

"require-corp"

When this value is used, fetching cross-origin resources requires the server's explicit permission through the CORS protocol or the `Cross-Origin-Resource-Policy` header.

An embedder policy consists of:

- A value, which is an embedder policy value\(^\text{855}\), initially "unsafe-none\(^\text{855}\)."
• A **reporting endpoint** string, initially the empty string.

• A **report only value**, which is an embedder policy value, initially "unsafe-none".

• A **report only reporting endpoint** string, initially the empty string.

The "coep" report type is a report type whose value is "coep". It is visible to ReportingObservers.

### 7.8.1 The headers

The `Cross-Origin-Embedder-Policy` and `Cross-Origin-Embedder-Policy-Report-Only` HTTP response header fields allow a server to declare an embedder policy for an environment settings object. These headers are structured headers whose values must be token.

The valid token values are the embedder policy values. The token may also have attached parameters; of these, the "report-to" parameter can have a valid URL string identifying an appropriate reporting endpoint.

<table>
<thead>
<tr>
<th><code>Cross-Origin-Embedder-Policy</code></th>
<th>Final embedder policy value</th>
</tr>
</thead>
<tbody>
<tr>
<td>No header delivered</td>
<td>&quot;unsafe-none&quot;</td>
</tr>
<tr>
<td>'require-corp'</td>
<td>&quot;require-corp&quot;</td>
</tr>
<tr>
<td>'unknown-value'</td>
<td>&quot;unsafe-none&quot;</td>
</tr>
<tr>
<td>'require-corp, unknown-value'</td>
<td>&quot;unsafe-none&quot;</td>
</tr>
<tr>
<td>'unknown-value, unknown-value'</td>
<td>&quot;unsafe-none&quot;</td>
</tr>
<tr>
<td>'unknown-value, require-corp'</td>
<td>&quot;unsafe-none&quot;</td>
</tr>
<tr>
<td>'require-corp, require-corp'</td>
<td>&quot;unsafe-none&quot;</td>
</tr>
</tbody>
</table>

*(The same applies to `Cross-Origin-Embedder-Policy-Report-Only`.)*

To obtain an embedder policy from a response and an environment:

1. Let `policy` be a new embedder policy.
2. If `environment` is a non-secure context, then return `policy`.
3. Let `parsedItem` be the result of getting a structured field value with `Cross-Origin-Embedder-Policy` and "item" from response's header list.
4. If `parsedItem` is non-null and `parsedItem[0]` is "require-corp":
   1. Set policy's value to "require-corp".
   2. If `parsedItem[1]"report-to"` exists, then set policy's endpoint to `parsedItem[1]"report-to"`.
5. Set `parsedItem` to the result of getting a structured field value with `Cross-Origin-Embedder-Policy-Report-Only` and "item" from response's header list.
6. If `parsedItem` is non-null and `parsedItem[0]` is "require-corp":
   1. Set policy's report-only value to "require-corp".
   2. If `parsedItem[1]"report-to"` exists, then set policy's report-only reporting endpoint to `parsedItem[1]"report-to"`.
7. Return `policy`. 
To check a navigation response's adherence to its embedder policy given a response, a browsing context target, and an environment environment:

1. If target is not a child browsing context, then return true.
2. Let responsePolicy be the result of obtaining an embedder policy from response and environment.
3. Let parentPolicy be target's container document's embedder policy.
4. If parentPolicy's report-only value is "require-corp" and responsePolicy's value is "unsafe-none", then queue a cross-origin embedder policy inheritance violation with response, "navigation", parentPolicy's report only reporting endpoint, "reporting", and target's container document's relevant settings object.
5. If parentPolicy's value is "unsafe-none" or responsePolicy's value is "require-corp", then return true.
7. Return false.

To check a global object's embedder policy given a WorkerGlobalScope, an environment settings object owner, and a response:

1. If workerGlobalScope is not a DedicatedWorkerGlobalScope object, then return true.
2. Let policy be workerGlobalScope's embedder policy.
3. Let ownerPolicy be owner's embedder policy.
4. If ownerPolicy's report-only value is "require-corp" and policy's value is "unsafe-none", then queue a cross-origin embedder policy inheritance violation with response, "worker initialization", owner's policy's report only reporting endpoint, "reporting", and owner.
5. If ownerPolicy's value is "unsafe-none" or policy's value is "require-corp", then return true.
7. Return false.

To queue a cross-origin embedder policy inheritance violation given a response, a string type, a string endpoint, a string disposition, and an environment settings object settings:

1. Let serialized be the result of serializing a response URL for reporting with response.
2. Let body be a new object containing the following properties:

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>type</td>
</tr>
<tr>
<td>blockedURL</td>
<td>serialized</td>
</tr>
<tr>
<td>disposition</td>
<td>disposition</td>
</tr>
</tbody>
</table>

3. Queue body as the "coep" report type for endpoint on settings.
7.9 Session history and navigation

7.9.1 Browsing sessions

A browsing session is ... See whatwg/html issue #4782 and whatwg/html issue #5350 for defining browsing session. It is roughly analogous to a top-level browsing context except that it cannot be replaced due to a `Cross-Origin-Opener-Policy` header or navigation.

A top-level browsing context has an associated browsing session which is a browsing session.

The browsing session of an environment settings object environment is the result of running these steps:

1. Assert: environment has a responsible document.
2. Return environment's responsible document's browsing context's top-level browsing context's browsing session.

7.9.2 The session history of browsing contexts

The sequence of Document's in a browsing context is its session history. Each browsing context, including child browsing contexts, has a distinct session history. A browsing context's session history consists of a flat list of session history entries.

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 getter steps are to return this's associated Document's History instance.

A session history entry is a struct with the following items:

- URL, a URL
- document, a Document or null
- serialized state, which is serialized state or null, initially null
- title, a string or null
- scroll restoration mode, a scroll restoration mode, initially "auto".
- scroll position data, which is scroll position data for the document's restorable scrollable regions.
- browsing context name, a browsing context name
- persisted user state, which is implementation-defined, initially null

Example

For example, some user agents might want to persist the values of form controls.
Serialized state is a serialization (via StructuredSerializeForStorage\(^{108}\)) of an object representing a user interface state. We sometimes informally refer to “state objects”, which are the objects representing user interface state supplied by the author, or alternately the objects created by deserializing (via StructuredDeserialize\(^{108}\)) serialized state.

Pages can add\(^{863}\) serialized state\(^{859}\) to the session history. These are then deserialized\(^{108}\) and returned to the script\(^{1267}\) when the user (or script) goes back in the history, thus enabling authors to use the “navigation” metaphor even in one-page applications.

A scroll restoration mode indicates whether the user agent should restore the persisted scroll position (if any) when traversing to an entry\(^{858}\). A scroll restoration mode is 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

Several contiguous entries in a session history can share the same document\(^{858}\). This can occur when the initial entry is reached via normal navigation\(^{875}\), and the following entry is added via history.pushState()\(^{863}\). Or it can occur via navigation to a fragment\(^{888}\).

User agents may discard\(^{813}\) the documents\(^{858}\) of entries with non-null documents\(^{858}\), as long as the following conditions are met:

- They must not discard the document\(^{858}\) of the current entry\(^{859}\).
- They must not discard any Document\(^{115}\) objects which are referenced from script.

Apart from these restrictions, this standard does not specify when user agents should discard an entry’s document\(^{858}\), versus keeping it cached.

User agents that persist the value of form controls are encouraged to also persist their directionality (the value of the element’s dir\(^{141}\) 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.

Note

Serialized state\(^{859}\) is intended to be used for two main purposes: first, storing a preparsed 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). Second, so that the author can store state that one wouldn’t store in the URL because it only applies to the current Document\(^{115}\) instance and it would have to be reconstructed if a new Document\(^{115}\) were opened.

An example of the latter would be something like keeping track of the precise coordinate from which a popup div\(^{238}\) 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.

Note

All entries that share the same document\(^{858}\) (and that are therefore merely different states of one particular document) are contiguous by definition.

User agents may discard\(^{813}\) the documents\(^{858}\) of entries with non-null documents\(^{858}\), as long as the following conditions are met:

- They must not discard the document\(^{858}\) of the current entry\(^{859}\).
- They must not discard any Document\(^{115}\) objects which are referenced from script.

Apart from these restrictions, this standard does not specify when user agents should discard an entry's document\(^{858}\), versus keeping it cached.

Note

Discarding\(^{883}\) a Document\(^{115}\) will set the corresponding document\(^{858}\) item of any session history entries\(^{858}\) to null. Subsequent navigations to those entries will result in the creation of a new Document\(^{115}\) object, and set the document\(^{858}\) item to it.

At any point, one of the entries in the session history is the current entry. This is the entry representing the active document\(^{813}\) of the browsing context\(^{813}\). Which entry is the current entry\(^{859}\) is changed by the algorithms defined in this specification, e.g., during session history traversal\(^{880}\).

859
The current entry is usually the initial entry created upon navigation. However, it can also be one of the contiguous entries that share the same document, as described above.

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.

7.9.3 The History interface

IDL

```idl
enum ScrollRestoration { "auto", "manual" };

[Exposed=Window]
interface History {
    readonly attribute unsigned long length;
    attribute ScrollRestoration scrollRestoration;
    readonly attribute any state;
    undefined go(optional long delta = 0);
    undefined back();
    undefined forward();
    undefined pushState(any data, DOMString title, optional USVString? url = null);
    undefined replaceState(any data, DOMString title, optional USVString? url = null);
}
```

For web developers (non-normative)

```javascript
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 serialized state, deserialized into an 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 increasing integers (1, 2, 3, etc).

Each History object has state, initially null.

The length getter steps are:

1. If this's associated Document is not fully active, then throw a "SecurityError" DOMException.
2. Return the number of entries in the top-level browsing context's joint session history.

The actual entries are not accessible from script.

The scrollRestoration getter steps are:

1. If this's associated Document is not fully active, then throw a "SecurityError" DOMException.
2. Return this's session history's current entry's scroll restoration mode.

The scrollRestoration setter steps are:

1. If this's associated Document is not fully active, then throw a "SecurityError" DOMException.
2. Set this's session history's current entry's scroll restoration mode to the given value.

The state getter steps are:

1. If this's associated Document is not fully active, then throw a "SecurityError" DOMException.
2. Return this's state.

The go(delta) method steps are:

1. Let document be this's associated Document.
2. If document is not fully active, then throw a "SecurityError" DOMException.
3. If delta is 0, then act as if the location.reload() method was called, and return.
4. Traverse the history by a delta with delta and document's browsing context.

The back() method steps are:

1. Let document be this's associated Document.
2. If document is not fully active, then throw a "SecurityError" DOMException.
3. Traverse the history by a delta with −1 and document's browsing context.

The forward() method steps are:

1. Let document be this's associated Document.
2. If document is not fully active, then throw a "SecurityError" DOMException.
3. Traverse the history by a delta with +1 and document's browsing context.

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 navigates from example.com to shop.example), transitions would otherwise have to involve cross-event-loop synchronization.

To traverse the history by a delta given delta and browsing context source browsing context, 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 return.
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 source browsing context is not allowed to navigate specified browsing context, then return.
5. If the specified browsing context's active document's unload algorithm is currently running, return.
6. Queue a global task on the history traversal task source given specified browsing context's active window to perform the following steps:
   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 return.
      2. Unload the active document of the specified browsing context.
      3. Traverse the history of the specified browsing context to the specified entry with explicitHistoryNavigation set to true.

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 with a delta equivalent to the action specified by the user and the browsing context being operated on.

The URL and history update steps, given a Document document, a URL newURL, an optional serialized state-or-null serializedData (default null), an optional string-or-null title (default null), and an optional boolean isPush (default false), are:

1. Let browsingContext be document's browsing context.
2. If isPush is true, then:
   1. Remove all the entries in browsingContext'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. Save persisted state to the current entry.

4. Add a session history entry entry to browsingContext's session history, after the current entry, with
   - newURL as the URL;
   - document as the document;
   - serializedData as the serialized state;
   - title as the title;
   - 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.

3. Otherwise:
   1. Let entry be browsingContext's session history's current entry.
   2. Set entry's URL to newURL.
   3. If serializedData is not null, then set entry's serialized state to serializedData.
   4. Set entry's title to title.
   5. Update entry so that it represents a GET request, if it currently represents a non-GET request (e.g. it was the result of a POST submission).

What does this mean? This is not a part of the definition of session history entry.

4. Set document's URL to newURL.

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.

The pushState(data, title, url) method steps are to run the shared history push/replace state steps given this, data, title; url, and true.

The replaceState(data, title, url) method steps are to run the shared history push/replace state steps given this, data, title; url, and false.

The shared history push/replace state steps, given a History history, a value data, a string title, a scalar value string-or-null url, and a boolean isPush, are:

1. Let document be history's associated Document.
2. If document is not fully active, then throw a "SecurityError" DOMException.
3. Optionally, return. (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.)
4. Let targetRealm be history's relevant Realm.
5. Let serializedData be StructuredSerializeForStorage(data). Rethrow any exceptions.
6. Let newURL be the session history's current entry's URL.
7. If url is not null, then:
   1. Parse url, relative to the relevant settings object of history.
   2. If that fails, then throw a "SecurityError" DOMException.
   3. Set newURL to the resulting URL record.
4. Compare `newURL` to `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.

5. If the `origin` of `newURL` is not same origin with the `origin` of `document`, and either the path or query components of the two URL records compared in the previous step differ, throw a "SecurityError" DOMException. (This prevents sandboxed content from spoofing other pages on the same origin.)

8. Run the URL and history update steps given `document` and `newURL`, with `serializedData` set to `serializedData`, `title` set to `title`, and `isPush` set to `isPush`.

9. Let state be `StructuredDeserialize(serializedData, targetRealm)`. If this throws an exception, catch it, ignore the exception, and set state to null.

10. Set `history's state` to state.

11. Set the `current entry's document's latest entry` to 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 implementation-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

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:

```html
<!DOCTYPE HTML>
<!-- this is https://example.com/line?x=5 -->
<html lang="en">
<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>
</html>
```

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:

```html
<!DOCTYPE HTML>
<!-- this starts off as https://example.com/line?x=5 -->
<html lang="en">
<title>Line Game - 5</title>
<p>You are at coordinate 5 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[1].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 naive 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

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.

```html
<!DOCTYPE HTML>
<html lang=en>
<title>Line</title>
<script>
  setInterval(function () { document.title = 'Line - ' + new Date(); }, 1000);
  var i = 1;
  function inc() {
    i = 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>
</html>
```

### Example

Most applications want to use the same scroll restoration mode value for all of their history entries. To achieve this they can 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.

```html
<head>
  <script>
    if ('scrollRestoration' in history)
      history.scrollRestoration = 'manual';
  </script>
</head>
```
7.9.4 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 objects 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.

7.9.5 The Location interface

Each Window object is associated with a unique instance of a Location object, allocated when the Window object is created.

⚠️ Warning!
The Location exotic object is defined through a mishmash of IDL, invocation of JavaScript internal methods post-creation, and overridden JavaScript internal methods. Coupled with its scary security policy, please take extra care while implementing this excrescence.

To create a Location object, run these steps:

1. Let location be a new Location platform object.
2. Let valueOf be location's relevant Realm.[Intrinsics].[[%Object.prototype.valueOf%]].
3. Perform ! location.[[DefineOwnProperty]]("valueOf", { [[Value]]: valueOf, [[Writable]]: false, [[Enumerable]]: false, [[Configurable]]: false }).
4. Perform ! location.[[DefineOwnProperty]](@@toPrimitive, { [[Value]]: undefined, [[Writable]]: false, [[Enumerable]]: false, [[Configurable]]: false }).
5. Set the value of the [[DefaultProperties]] internal slot of location to location.[[OwnPropertyKeys]]().
6. Return location.

Note
The addition of valueOf and @@toPrimitive own data properties, as well as the fact that all of Location's IDL attributes are marked [LegacyUnforgeable], is required by legacy code that consulted the Location interface, or stringified it, to determine the document URL, and then used it in a security-sensitive way. In particular, the valueOf, @@toPrimitive, and [LegacyUnforgeable] stringifier mitigations ensure that code such as foo[location] = bar or location + "" cannot be misdirected.

For web developers (non-normative)

```
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 `Document` object's `location` attribute's getter must return this `Document` object's relevant global object's `Location` object, if this `Document` object is fully active, and null otherwise.

The `Window` object's `location` attribute's getter must return this `Window` object's `Location` 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.

```
[Exposed=Window]
interface Location { // but see also additional creation steps and overridden internal methods
  [LegacyUnforgeable] stringifier attribute USVString href;
  [LegacyUnforgeable] readonly attribute USVString origin;
  [LegacyUnforgeable] attribute USVString protocol;
  [LegacyUnforgeable] attribute USVString host;
  [LegacyUnforgeable] attribute USVString hostname;
  [LegacyUnforgeable] attribute USVString port;
  [LegacyUnforgeable] attribute USVString pathname;
  [LegacyUnforgeable] attribute USVString search;
  [LegacyUnforgeable] attribute USVString hash;

  [LegacyUnforgeable] undefined assign(USVString url);
  [LegacyUnforgeable] undefined replace(USVString url);
  [LegacyUnforgeable] undefined reload();

  [LegacyUnforgeable, SameObject] readonly attribute DOMStringList ancestorOrigins;
};
```

For web developers (non-normative)

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 a DOMStringList object listing 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 relevant global object's browsing context's active document, if this Location object's relevant global object's browsing context is non-null, and null otherwise.

A Location object has an associated url, which is this Location object's relevant Document's URL, if this Location object's relevant Document is non-null, and about:blank otherwise.

A Location object has an associated ancestor origins list. When a Location object is created, its ancestor origins list must be set to a DOMStringList object whose associated list is the list of strings that the following steps would produce:

1. Let output be a new list of strings.
2. Let current be the browsing context of the Document with which this 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 serialization of current's active document's origin to output.
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. Let historyHandling be "replace".
2. If any of the following conditions are met, then set historyHandling to "default":
   - 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's isTrusted attribute is true, or
   - In the task, in which the algorithm is running, the event listener for a click event, whose isTrusted attribute is true, is being handled.
3. Location-object navigate, given url and historyHandling.

To Location-object navigate, given a url and historyHandling:

1. Let browsingContext be the current global object's browsing context.
2. Let sourceBrowsingContext be the incumbent global object's browsing context.
3. If browsingContext's session history contains only one Document, and that was the about:blank Document created when the browsing context was created, then set historyHandling to "replace".
4. Navigate browsingContext to url, with exceptionsEnabled set to true, historyHandling set to historyHandling, and the source browsing context set to sourceBrowsingContext.
The `href` attribute's getter must run these steps:

1. If this `Location` object's relevant `Document` is non-null and its `origin` is not `same-origin-domain` with the `entry settings object`'s `origin`, then throw a `SecurityError` DOMException.
2. Return this `Location` object's `url`, serialized.

The `href` attribute's setter must run these steps:

1. If this `Location` object's relevant `Document` is null, then return.
2. Parse the given value relative to the `entry settings object`. If that failed, throw a `TypeError` exception.
3. `Location-object-setter navigate` given the resulting URL record.

The `origin` attribute's getter must run these steps:

1. If this `Location` object's relevant `Document` is non-null and its `origin` is not `same-origin-domain` with the `entry settings object`'s `origin`, then throw a `SecurityError` DOMException.
2. Return the serialization of this `Location` object's `url`'s `origin`.

The `protocol` attribute's getter must run these steps:

1. If this `Location` object's relevant `Document` is non-null and its `origin` is not `same-origin-domain` with the `entry settings object`'s `origin`, then throw a `SecurityError` DOMException.
2. Return this `Location` object's `url`'s `scheme`, followed by `:`.

The `protocol` attribute's setter must run these steps:

1. If this `Location` object's relevant `Document` is null, then return.
2. 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.
3. Let `copyURL` be a copy of this `Location` object's `url`.
4. 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`.

```
Note
Because the URL parser ignores multiple consecutive colons, providing a value of "https:" (or even "https::") is the same as providing a value of "https".
```

5. If `possibleFailure` is failure, then throw a `SyntaxError` DOMException.
6. If `copyURL`'s `scheme` is not an `HTTP(S)` scheme, then terminate these steps.
7. `Location-object-setter navigate` to `copyURL`.

The `host` attribute's getter must run these steps:

1. If this `Location` object's relevant `Document` is non-null and its `origin` is not `same-origin-domain` with the `entry settings object`'s `origin`, then throw a `SecurityError` DOMException.
2. Let `url` be this `Location` object's `url`.
3. If `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.

---

Note

The `href` attribute setter intentionally has no security check.

---

Because the URL parser ignores multiple consecutive colons, providing a value of "https:" (or even "https::") is the same as providing a value of "https".
The `host` attribute's setter must run these steps:

1. If this `Location` object's relevant Document is null, then return.
2. 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.
3. Let `copyURL` be a copy of this `Location` object's URL.
4. If `copyURL`'s `cannot-be-a-base-URL` is true, then return.
5. Basic URL parse the given value, with `copyURL` as `url` and `host state` as state override.
6. `Location-object-setter navigate` to `copyURL`.

The `hostname` attribute's getter must run these steps:

1. If this `Location` object's relevant Document is non-null and its origin is not same origin-domain with the entry settings object's origin, then throw a "SecurityError" DOMException.
2. If this `Location` object's `url`'s `host` is null, return the empty string.
3. Return this `Location` object's `url`'s `host`, serialized.

The `hostname` attribute's setter must run these steps:

1. If this `Location` object's relevant Document is null, then return.
2. 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.
3. Let `copyURL` be a copy of this `Location` object's `url`.
4. If `copyURL`'s `cannot-be-a-base-URL` is true, then return.
5. Basic URL parse the given value, with `copyURL` as `url` and `hostname state` as state override.
6. `Location-object-setter navigate` to `copyURL`.

The `port` attribute's getter must run these steps:

1. If this `Location` object's relevant Document is non-null and its origin is not same origin-domain with the entry settings object's origin, then throw a "SecurityError" DOMException.
2. If this `Location` object's `url`'s `port` is null, return the empty string.
3. Return this `Location` object's `url`'s `port`, serialized.

The `port` attribute's setter must run these steps:

1. If this `Location` object's relevant Document is null, then return.
2. 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.
3. Let `copyURL` be a copy of this `Location` object's `url`.
4. If `copyURL` cannot have a username/password/port, then return.
5. If the given value is the empty string, then set `copyURL`'s `port` to null.
6. Otherwise, basic URL parse the given value, with `copyURL` as `url` and `port state` as state override.
7. `Location-object-setter navigate` to `copyURL`.

The `pathname` attribute's getter must run these steps:

1. If this `Location` object's relevant Document is non-null and its origin is not same origin-domain with the entry settings object's origin, then throw a "SecurityError" DOMException.
2. Let url be this Location object's url.
3. If url's cannot-be-a-base-URL is true, then return url's path[0].
4. If url's path is empty, then return the empty string.
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. If this Location object's relevant Document is null, then return.
2. 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.
3. Let copyURL be a copy of this Location object's url.
4. If copyURL's cannot-be-a-base-URL is true, then return.
5. Set copyURL's path to the empty list.
6. Basic URL parse the given value, with copyURL as url and path start state as state override.
7. Location-object-setter navigate to copyURL.

The search attribute's getter must run these steps:

1. If this Location object's relevant Document is non-null and its origin is not same origin-domain with the entry settings object's origin, then throw a "SecurityError" DOMException.
2. If this Location object's url's query is either null or the empty string, return the empty string.
3. Return "?", followed by this Location object's url's query.

The search attribute's setter must run these steps:

1. If this Location object's relevant Document is null, then return.
2. 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.
3. Let copyURL be a copy of this Location object's url.
4. If the given value is the empty string, set copyURL'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 copyURL's query to the empty string.
   3. Basic URL parse input, with null, the relevant Document's document's character encoding, copyURL as url, and query state as state override.
7. Location-object-setter navigate to copyURL.

The hash attribute's getter must run these steps:

1. If this Location object's relevant Document is non-null and its origin is not same origin-domain with the entry settings object's origin, then throw a "SecurityError" DOMException.
2. If this Location object's url's fragment is either null or the empty string, return the empty string.
3. Return "#", followed by this Location object's url's fragment.

The hash attribute's setter must run these steps:

1. If this Location object's relevant Document is null, then return.
2. If this Location object's relevant Document's origin is not same origin-domain with the entry settings object's
Let `copyURL` be a copy of this `Location` object's `url`.

Let `input` be the given value with a single leading "#" removed, if any.

Set `copyURL`'s fragment to the empty string.

Basic URL parse `input`, with `copyURL` as `url` and fragment state as state override.

Location-object-setter navigate to `copyURL`.

When the `assign(url)` method is invoked, the user agent must run the following steps:

1. If this `Location` object's relevant Document is null, then return.
2. 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.
3. Parse `url` relative to the entry settings object. If that failed, throw a "SyntaxError" DOMException.
4. Location-object navigate given the resulting URL record and "default".

When the `replace(url)` method is invoked, the user agent must run the following steps:

1. If this `Location` object's relevant Document is null, then return.
2. Parse `url` relative to the entry settings object. If that failed, throw a "SyntaxError" DOMException.
3. Location-object navigate given the resulting URL record and "replace".

Note
Unlike the equivalent API for the `a` and `area` elements, the `hash` attribute's setter does not special case the empty string to remain compatible with deployed scripts.

When the `reload()` method is invoked, the user agent must run the appropriate steps from the following list:

↪ If this `Location` object's relevant Document is null
   Return.

↪ If this `Location` object's relevant Document's `origin` is not same origin-domain with the entry settings object's `origin`
   Throw a "SecurityError" DOMException.

↪ 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 return.

↪ If the `browsing context`'s active document is an iframe srcdoc document
   Reprocess the iframe attributes of the browsing context's container.

↪ Otherwise
   Navigate to this `Location` object's relevant Document's `URL`, with exceptionsEnabled set to true, `historyHandling` set to "reload", and the source browsing context set to the browsing context being navigated.

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 historyHandling set to "reload". 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.

The `ancestorOrigins` attribute's getter must run these steps:

1. If this `Location` object's `relevant Document` is null, then return an empty list.
2. 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`.
3. Otherwise, return this `Location` object's `ancestor origins list`.

⚠️ Warning!

The details of how the `ancestorOrigins` attribute works are still controversial and might change. See issue #1918 for more information.

As explained earlier, the `Location` exotic object requires additional logic beyond IDL for security purposes. The `Location` object must use the ordinary internal methods except where it is explicitly specified otherwise below.

Also, every `Location` object has a `[[DefaultProperties]]` internal slot representing its own properties at time of its creation.

7.9.5.1 `[[GetPrototypeOf]] ( )` §3

1. If ! `IsPlatformObjectSameOrigin` (this) is true, then return ! `OrdinaryGetPrototypeOf`(this).
2. Return null.

7.9.5.2 `[[SetPrototypeOf]] ( V )` §3

1. Return ! `ImmutablePrototype`(this, V).

7.9.5.3 `[[IsExtensible]] ( )` §3

1. Return true.

7.9.5.4 `[[PreventExtensions]] ( )` §3

1. Return false.

7.9.5.5 `[[GetOwnProperty]] ( P )` §3

1. If ! `IsPlatformObjectSameOrigin` (this) is true, then:
   1. Let desc be ! `OrdinaryGetOwnProperty`(this, P).
   2. If the value of the `[[DefaultProperties]]` internal slot of this contains P, then set desc.[[Configurable]] to true.
   3. Return desc.
2. Let property be ! `CrossOriginGetOwnPropertyHelper`(this, P).
3. If property is not undefined, then return property.
7.9.5.6 [[DefineOwnProperty]] (P, Desc) §p87
1. If !IsPlatformObjectSameOrigin\textsuperscript{p825}(this) is true, then:
   1. If the value of the [[DefaultProperties]]\textsuperscript{p873} internal slot of this contains P, then return false.
   2. Return ?OrdinaryDefineOwnProperty(this, P, Desc).
2. Throw a "SecurityError"\ DOMException.

7.9.5.7 [[Get]] (P, Receiver) §p87
1. If !IsPlatformObjectSameOrigin\textsuperscript{p825}(this) is true, then return ?OrdinaryGet(this, P, Receiver).
2. Return ?CrossOriginGet\textsuperscript{p826}(this, P, Receiver).

7.9.5.8 [[Set]] (P, V, Receiver) §p87
1. If !IsPlatformObjectSameOrigin\textsuperscript{p825}(this) is true, then return ?OrdinarySet(this, P, Receiver).
2. Return ?CrossOriginSet\textsuperscript{p826}(this, P, V, Receiver).

7.9.5.9 [[Delete]] (P) §p87
1. If !IsPlatformObjectSameOrigin\textsuperscript{p825}(this) is true, then return ?OrdinaryDelete(this, P).
2. Throw a "SecurityError"\ DOMException.

7.9.5.10 [[OwnPropertyKeys]] () §p87
1. If !IsPlatformObjectSameOrigin\textsuperscript{p825}(this) is true, then return !OrdinaryOwnPropertyKeys(this).
2. Return !CrossOriginOwnPropertyKeys\textsuperscript{p827}(this).

7.10 Browsing the web §p87
7.10.1 Navigating across documents §p87
Certain actions cause the browsing context\textsuperscript{p811} to navigate\textsuperscript{p875} 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
For example, following a hyperlink\textsuperscript{p291}, form submission\textsuperscript{p596}, and the window.open()\textsuperscript{p388} and location.assign()\textsuperscript{p872} 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\textsuperscript{p363} needs to know the data it is to use.

Much of the navigation process is concerned with determining how to create a new Document\textsuperscript{p815}, which ultimately happens in the create and initialize a Document object\textsuperscript{p882} algorithm. The parameters to this algorithm are tracked via a navigation params struct, which has the following items:
null or a request that started the navigation

null or a response that ultimately was navigated to (potentially a network error)

an origin to use for the new Document

a sandboxing flag set to impose on the new Document

da cross-origin opener policy to use for the new Document

da cross-origin opener policy enforcement result, used for reporting and potentially for causing a browsing context group switch

null or an environment reserved for the new Document

to the browsing context to be navigated (or discarded, if a browsing context group switch occurs)

a history handling behavior

Once a navigation params struct is created, this standard does not mutate any of its items. They are only passed onward to other algorithms.

A regular navigation which adds a new entry to the session history.

A navigation to an existing session history entry to recreate that entry's document, which was previously discarded.

A navigation intended to reload the current page and replace the current session history entry.

A non-reload navigation that will replace the current session history entry.

After Document creation, the session history gets updated. A history handling behavior is used to track the desired type of session history update throughout the navigation process. It is one of the following:

"default"

"entry update"

"reload"

"replace"

Navigation always involves source browsing context, which is the browsing context which was responsible for starting the navigation.

As explained in issue #1130 the use of a browsing context as source might not be the correct architecture.

To navigate a browsing context browsingContext to a resource resource, with an optional boolean exceptionsEnabled (default false), an optional history handling behavior historyHandling (default "default"), and an optional string navigationType (default "other"):

1. If resource is a URL, then set resource to a new request whose url is resource.
2. If resource is a request and historyHandling is "reload", then set resource's reload-navigation flag.
3. If the source browsing context is not allowed to navigate browsingContext, then:
   1. If exceptionsEnabled is given and is true, then throw a "SecurityError" DOMException.
   2. Otherwise, the user agent may instead offer to open resource in a new top-level browsing context or in the top-
4. If there is a preexisting attempt to navigate `browsingContext`, and the `source browsing context` is the same as `browsingContext`, and that attempt is currently running the `upload a document` algorithm, then return without affecting the preexisting attempt to navigate `browsingContext`.

5. If the `prompt to unload` algorithm is being run for the `active document` of `browsingContext`, then return without affecting the `prompt to unload` algorithm.

6. If `historyHandling` is not `reload`, `resource` is a `request`, `resource`'s `url` equals `browsingContext`'s `active document`'s `URL`, with `exclude fragments` set to true, and `resource`'s `url`'s `fragment` is non-null, then:
   1. Navigate to a fragment given `browsingContext`, `resource`'s `url`, and `historyHandling`.
   2. Return.


8. Let `incumbentNavigationOrigin` be the `origin` of the `incumbent settings object`, or if no `script` was involved, the `origin` of the `node document` of the element that initiated the `navigation`.

9. Cancel any preexisting but not yet `mature` attempt to navigate `browsingContext`, 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.)

10. Prompt to unload the `active document` of `browsingContext`. If the user refused to allow the document to be unloaded, then return.

    If this instance of the `navigation` algorithm gets canceled while this step is running, the `prompt to unload` algorithm must nonetheless be run to completion.

11. Abort the `active document` of `browsingContext`.

12. If `browsingContext` is a `child browsing context`, then put it in the `delaying load events mode`.

    The user agent must take this `child 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. Let `sandboxFlags` be the result of `determining the creation sandboxing flags` given `browsingContext` and `browsingContext`'s `container`.

14. Return to whatever algorithm invoked the navigation steps and continue running these steps in parallel.

15. This is the step that attempts to obtain `resource`, if necessary. Jump to the first appropriate substep:

   If `resource` is a `response`:
   1. Assert: `browsingContext` is not a `top-level browsing context`.
   2. Let `finalSandboxFlags` be the `union` of `browsingContext`'s `sandboxing flags` and `resource`'s `forced sandboxing flag set`.
   4. Let `coop` be a new `cross-origin opener policy`.
   5. Let `coopEnforcementResult` be a new `cross-origin opener policy enforcement result` whose `needs a browsing context group switch` is false, `would need a browsing context group switch due to report-only` is false, `url` is `resource`'s `url`, `origin` is `responseOrigin`, the `cross-origin opener policy`, is `coop`, and `current context is...`
Let `browsing context` whose `url`'s `scheme` is false.

6. Let `navigationParams` be a new `navigation params` whose `request` is null, `response` is resource, `origin` is `responseOrigin`, `final sandboxing flag set` is `finalSandboxFlags`, `cross-origin opener policy` is `coo`, `COOP enforcement result` is `cooopEnforcementResult`, `reserved environment` is null, `browsing context` is `browsingContext`, and `history handling` is `historyHandling`.

7. Run `process a navigate response` with `navigationType`, the `source browsing context`, and `navigationParams`.

### If `resource` is a `request` whose `url`'s `scheme` is "javascript"

Queue a global task on the DOM manipulation task source given `browsingContext`'s active window to run these steps:

1. Let `response` be the result of executing a `javascript: URL` `request` given `resource`, the `source browsing context`, and `browsingContext`.

2. Let `finalSandboxFlags` be the `union` of `browsingContext`'s `sandbox flags` and `response`'s `forced sandboxing flag set`.

3. Let `cooopEnforcementResult` be a new `cross-origin opener policy enforcement result` whose `needs a browsing context group switch` is false, would need a browsing context group switch due to report-only is false, `url` is `resource`'s `url`, `origin` is `activeDocumentNavigationOrigin`, `cross-origin opener policy` is `browsingContext`'s `active document`'s `cross-origin opener policy`, and `current context is navigation source` is false.

4. Let `navigationParams` be a new `navigation params` whose `request` is `resource`, `response` is `response`, `origin` is `activeDocumentNavigationOrigin`, `final sandboxing flag set` is `finalSandboxFlags`, `cross-origin opener policy` is `browsingContext`'s `active document`'s `cross-origin opener policy`, `COOP enforcement result` is `cooopEnforcementResult`, `reserved environment` is null, `browsing context` is `browsingContext`, and `history handling` is `historyHandling`.

5. Run `process a navigate response` with `navigationType`, the `source browsing context`, and `navigationParams`.

### Example

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 nested `browsingContext` when the `iframe` is being set up. Once evaluated, its return value (if it was a string) would replace that `browsing context`'s `active document`, thus also changing the corresponding `Window` object.

### If `resource` is a `request` whose `url`'s `scheme` is a `fetch scheme`

Run `process a navigate fetch` given `resource`, the `source browsing context`, `browsingContext`, `navigationType`, `sandboxFlags`, `incumbentNavigationOrigin`, `activeDocumentNavigationOrigin`, and `historyHandling`.

Otherwise, `resource` is a `request` whose `url`'s `scheme` is neither "javascript" nor a `fetch scheme`

Run `process a navigate URL scheme` given resource's `url` and `browsingContext`.

To `process a navigate fetch`, given a `request` request, two `browsing contexts` source `browsingContext` and `browsingContext`, a string `navigationType`, a `sandboxing flag set` `sandboxFlags`, two `origins` `incumbentNavigationOrigin` and `activeDocumentNavigationOrigin`, and a `history handling behavior` `historyHandling`:

1. Let response be null.

2. Set `request`'s `client` to `sourceBrowsingContext`'s `active document`'s `relevant settings object`, `destination` to "document", `mode` to "navigate", `credentials mode` to "include", use-URL-credentials flag, `redirect mode` to "manual", and replaces client id to `browsingContext`'s `active document`'s `relevant settings object`'s `id`.

3. If `browsingContext`'s `container` is non-null:
   1. If the `browsingContext`'s `container` has a `browsing context scope origin`, then set `request`'s `origin` to that `browsing context scope origin`.
   2. Set `request`'s `destination` to `browsingContext`'s `container`'s local name.
4. Let `reservedEnvironment` be null.

5. Let `responseOrigin` be null.

6. Let `currentContextIsSource` be the result of whether `browsingContext`'s active document is same origin with source's active document.

7. Let `coopEnforcementResult` be a new cross-origin opener policy enforcement result whose needs a browsing context group switch is false, would need a browsing context group switch due to report-only is false, url is `browsingContext`'s active document's url, origin is `browsingContext`'s active document's origin, cross-origin opener policy is `browsingContext`'s active document's cross-origin opener policy, and current context is navigation source is `currentContextIsSource`.

8. Let `finalSandboxFlags` be an empty sandboxing flag set.

9. Let `locationURL` be null.

10. Let `currentURL` be `request`'s current URL.

11. While true:

   1. If `locationURL` is non-null, then set `currentURL` to `locationURL`.

   2. If `reservedEnvironment` is not null and `currentURL`'s origin is not the same as reservedEnvironment's creation URL's origin, then:

      1. Run the environment discarding steps for `reservedEnvironment`.

      2. Set `reservedEnvironment` to null.

   3. If `reservedEnvironment` is null, then:

      1. Let `topLevelCreationURL` be `currentURL`.

      2. Let `topLevelOrigin` be null.

      3. If `browsingContext` is not a top-level browsing context, then:

         1. Let `parentEnvironment` be `browsingContext`'s container's relevant settings object.

         2. Set `topLevelCreationURL` to `parentEnvironment`'s top-level creation URL and `topLevelOrigin` to `parentEnvironment`'s top-level origin.

     4. Set `reservedEnvironment` to a new environment whose id is a unique opaque string, target browsing context is `browsingContext`, creation URL is `currentURL`, top-level creation URL is `topLevelCreationURL`, and top-level origin is `topLevelOrigin`.

     

Note
The created environment's active service worker is set in the Handle Fetch algorithm during the fetch if the request URL matches a service worker registration. [SW]

4. Set `request`'s reserved client to `reservedEnvironment`.

5. If the result of Should navigation request of type be blocked by Content Security Policy? given `request` and navigationType is "Blocked", then set response to a network error and break. [CSP]

6. Otherwise:

   1. If `response` is null, fetch `request`.

   2. Otherwise, perform HTTP-redirect fetch using `request` and `response`.

   3. Wait for the task on the networking task source to process `response` and set `response` to the result.

   4. Set `finalSandboxFlags` to the union of `browsingContext`'s sandboxing flags and response's forced sandboxing flag set.

6. If `browsingContext` is a top-level browsing context\(^{6816}\), then:

   1. Set `responseCOOP` to the result of obtaining a cross-origin opener policy\(^{6848}\) given response and reservedEnvironment.

   2. If `sandboxFlags` is not empty and `responseCOOP`'s value\(^{6847}\) is not "unsafe-none\(^{6847}\)", then set `response` to an appropriate network error and break.

   ```plaintext
   Note
   This results in a network error as one cannot simultaneously provide a clean slate to a response using cross-origin opener policy and sandbox the result of navigating to that response.
   ```

   3. Set `coopEnforcementResult` to the result of enforcing the response's cross-origin opener policy\(^{6856}\) given `browsingContext`, request's `url`, responseOrigin, `responseCOOP`, `coopEnforcementResult` and request's `referrer`.

   7. If `response` is not a network error, `browsingContext` is a child browsing context\(^{6816}\), and the result of performing a cross-origin resource policy check with `browsingContext`'s container document\(^{6816}\)'s origin, `browsingContext`'s container document\(^{6816}\)'s relevant settings object\(^{9319}\), request's destination, response, and true is blocked, then set `response` to a network error and break.

   ```plaintext
   Note
   Here we're running the cross-origin resource policy check against the parent browsing context\(^{6816}\) rather than sourceBrowsingContext. This is because we care about the same-originness of the embedded content against the parent context, not the navigation source.
   ```

8. Set `locationURL` to `response`'s location URL.

9. If `locationURL` is not a URL whose scheme is an HTTP(S) scheme, then break.

   ```plaintext
   Note
   Navigation handles redirects manually as navigation is the only place in the web platform that cares for redirects to mailto: URLs and such.
   ```

12. If `locationURL` is failure, then set `response` to a network error.

13. Otherwise, if `locationURL` is a URL whose scheme is "blob", "file", "filesystem", or "javascript", then set `response` to a network error.

14. Otherwise, if `locationURL` is a URL whose scheme is a fetch scheme, then run process a navigate fetch\(^{6877}\) with a new request whose `url` is `locationURL`, `sourceBrowsingContext`, `browsingContext`, navigationType, `sandboxFlags`, incumbentNavigationOrigin, activeDocumentNavigationOrigin, and historyHandling, and return.

15. Otherwise, if `locationURL` is a URL, run the process a navigate URL scheme\(^{6881}\) given `locationURL` and `browsingContext`, and return.

16. Let `navigationParams` be a new navigation params\(^{6874}\) whose `request`\(^{6875}\) is `request`, response\(^{6875}\) is `response`, origin\(^{6875}\) is `origin`, final sandboxing flag set\(^ {6875}\) is `finalSandboxFlags`, cross-origin opener policy\(^{6875}\) is `responseCOOP`, COOP enforcement result\(^{6875}\) is `coopEnforcementResult`, reserved environment\(^{6875}\) is `reservedEnvironment`, browsing context\(^{6875}\) is `browsingContext`, and history handling\(^{6875}\) is `historyHandling`.

17. Run process a navigate response\(^{6879}\) with navigationType, the source browsing context\(^{6875}\), and `navigationParams`.

To process a navigate response, given a string navigationType, a browsing context\(^{6813}\) source, and a navigation params\(^{6874}\) `navigationParams`:

1. Let `response` be `navigationParams`'s response\(^{6875}\).

2. Let `browsingContext` be `navigationParams`'s browsing context\(^ {6875}\).

3. Let `failure` be false.

4. If `response` is a network error, then set `failure` to true.

Otherwise, if the result of Should navigation response to navigation request of type in target be blocked by Content Security...
Policy? given navigationParams's request, response, navType, and browsingContext is "Blocked", then set failure to true. [CSP] Otherwise, if navigationParams's reserved environment is non-null and the result of checking a navigation response's adherence to its embedder policy given response, browsingContext, and navigationParams's reserved environment is false, then set failure to true.

Otherwise, if the result of checking a navigation response's adherence to `X-Frame-Options` given response, browsingContext, and navigationParams's origin is false, then set failure to true.

5. If failure is true, then:
   1. Display the inline content with an appropriate error shown to the user given browsingContext.
   2. Run the environment discarding steps for navigationParams's reserved environment.
   3. Return.

6. If response's status is 204 or 205, then return.

7. If response has a `Content-Disposition` header specifying the attachment disposition type, then:
   1. If the result of running the allowed to download given source and browsingContext is true, then handle response as a download.
   2. Return.

8. Let type be the computed type of response.

9. 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 response as described there:

   ↪ an HTML MIME type
   Follow the steps given in the HTML document section providing navigationParams. Once the steps have completed, return.

   ↪ an XML MIME type that is not an explicitly supported XML MIME type
   Follow the steps given in the XML document section providing navigationParams and type. Once the steps have completed, return.

   ↪ a JavaScript MIME type
   ↪ a JSON MIME type that is not an explicitly supported JSON MIME type
   ↪ "text/css"
   ↪ "text/plain"
   ↪ "text/vtt"
   Follow the steps given in the plain text file section providing navigationParams and type. Once the steps have completed, return.

   ↪ multipart/x-mixed-replace
   Follow the steps given in the multipart/x-mixed-replace section providing navigationParams. Once the steps have completed, return.

   ↪ A supported image, video, or audio type
   Follow the steps given in the media section providing navigationParams and type. Once the steps have completed, return.

   ↪ A type that will use an external application to render the content in browsingContext
   Follow the steps given in the plugin section providing navigationParams and type. Once the steps have completed, return.

An explicitly supported XML MIME type is an XML MIME type for which the user agent is configured to use an external application to render the content (either a plugin rendering directly in browsingContext, 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 [2197] MIME type), or one for which the user agent has a dedicated handler.

An explicitly supported JSON MIME type is a JSON MIME type for which the user agent is configured to use an external application to render the content (either a plugin rendering directly in browsingContext, or a separate application), or for which the user agent has dedicated processing rules, or one for which the user agent has a dedicated handler.

10. 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 or an error message because the specified type is not supported, then display the inline content given browsingContext, and then return.

11. Otherwise, the document's type is such that the resource will not affect browsingContext, 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.

To process a navigate URL scheme, given a URL url and a browsing context browsingContext, run these steps:

1. If url is to be handled using a mechanism that does not affect browsingContext, e.g., because url's scheme is handled externally, then proceed with that mechanism instead.

2. Otherwise, url 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. Display the inline content given browsingContext.

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 was invoked when source browsing context's active window does not have transient activation, the user agent should not invoke the external software package without prior user confirmation.

To execute a javascript: URL request, given a request request and two browsing contexts source and browsingContext, run these steps:

1. Let response be a response whose status is 204.

2. If both of the following are true:
   ◦ source's active document's origin is same origin with browsingContext's active document's origin.
   ◦ The result of Should navigation request of type be blocked by Content Security Policy? given request and "other" is "Allowed".

   As explained in issue #2591 this step does not work and presents a security issue.

   The result of Should navigation request of type be blocked by Content Security Policy? given request and "other" is "Allowed". [CSP] then:

   1. Let urlString be the result of running the URL serializer on request's url.
   2. Let encodedScriptSource be the result of removing the leading "javascript:" from urlString.
   3. Let scriptSource be the UTF-8 decoding of the percent-decoding of encodedScriptSource.
   4. Append browsingContext's active document's URL to request's URL list.
   5. Let settings be browsingContext's active document's relevant settings object.
   6. Let baseURL be settings's API base URL.
7. Let script be the result of creating a classic script given scriptSource, settings, baseURL, and the default classic script fetch options.

8. Let evaluationStatus be the result of running the classic script script.

9. Let result be undefined if evaluationStatus is an abrupt completion or evaluationStatus.[Value] is empty, or evaluationStatus.[Value] otherwise.

10. If Type(result) is String, then set response to a response whose header list consists of `Content-Type`/`text/html` and `Referrer-Policy`/settings's referrer policy, and whose body is result.

   The exact conversion between the string result and the bytes that comprise a response body is not yet specified, pending further investigation into user agent behavior. See issue #1129.

3. Return response.

In addition to the specific issues linked above, javascript: URLs have a dedicated label on the issue tracker documenting various problems with their specification.

Some of the sections below, to which the above algorithm defers in certain cases, use the following steps to create and initialize a Document object, given a type, content type, and navigation params:

1. Let browsingContext be the result of obtaining a browsing context to use for a navigation response given navigationParams's browsing context, navigationParams's final sandboxing flag set, navigationParams's cross-origin opener policy, and navigationParams's COOP enforcement result.

2. Let permissionsPolicy be the result of creating a permissions policy from a response given browsingContext, navigationParams's origin, and navigationParams's response. [PERMISSIONSPOLICY]

   The creating a permissions policy from a response algorithm makes use of the passed origin. If document.domain has been used for browsingContext's container document, then its origin cannot be same origin-domain with the passed origin, because these steps run before the document is created, so it cannot itself yet have used document.domain. Note that this means that Permissions Policy checks are less permissive compared to doing a same origin check instead.

   See below for some examples of this in action.

3. Let creationURL be navigationParams's response URL.

4. If navigationParams's request is non-null, then set creationURL to navigationParams's request's current URL.

5. If browsingContext's only entry in its session history is the initial about:blank Document that was added when that browsing context was created, and navigationParams's history handling is replace, and that Document's origin is same origin-domain with navigationParams's origin, then do nothing.

6. Otherwise:

   1. Let oacHeader be the result of getting a structured field value given `Origin-Agent-Cluster` and "item" from response's header list.

   2. Let requestsOAC be true if oacHeader is not null and oacHeader[0] is the boolean true; otherwise false.

   3. If reservedEnvironment is a non-secure context, then set requestsOAC to false.

   4. Let agent be the result of obtaining a similar-origin window agent given navigationParams's origin, browsingContext's group, and requestsOAC.

   5. Let realm execution context be the result of creating a new JavaScript realm given agent and the following customizations:

      - For the global object, create a new Window object.

      - For the global this binding, use browsingContext's WindowProxy object.
6. Let \( \text{topLevelCreationURL} \) be \( \text{creationURL} \).

7. Let \( \text{topLevelOrigin} \) be \( \text{navigationParams}' \text{ origin} \).

8. If \( \text{browsingContext} \) is not a \( \text{top-level browsing context} \), then:
   1. Let \( \text{parentEnvironment} \) be \( \text{browsingContext}' \text{ container}' \text{ relevant settings object} \).
   2. Set \( \text{topLevelCreationURL} \) to \( \text{parentEnvironment}' \text{ top-level creation URL} \).
   3. Set \( \text{topLevelOrigin} \) to \( \text{parentEnvironment}' \text{ top-level origin} \).

9. Set up a window environment settings object with \( \text{creationURL} \), realm execution context, \( \text{navigationParams}' \text{ reserved environment} \), \( \text{topLevelCreationURL} \), and \( \text{topLevelOrigin} \).

7. Let \( \text{document} \) be a new \( \text{Document} \), whose type is \( \text{type} \), content type is \( \text{contentType} \), origin is \( \text{navigationParams}' \text{ origin} \), permissions policy is \( \text{permissionsPolicy} \), active sandboxing flag set is \( \text{navigationParams}' \text{ final sandboxing flag set} \), and cross-origin opener policy is \( \text{navigationParams}' \text{ cross-origin opener policy} \).

8. Set \( \text{document}' \text{ URL} \) to \( \text{creationURL} \).

9. Set \( \text{document}' \text{ referrer policy} \) to the result of parsing the \`Referrer-Policy\' header of \( \text{navigationParams}' \text{ response} \).

10. If \( \text{navigationParams}' \text{ reserved environment} \) is non-null, then set \( \text{document}' \text{ embedder policy} \) to the result of obtaining an embedder policy given \( \text{navigationParams}' \text{ response} \) and \( \text{navigationParams}' \text{ reserved environment} \). Otherwise, set it to "unsafe-none".

11. Initialize a \( \text{Document}' \text{ CSP list} \) given \( \text{document}, \text{navigationParams}' \text{ response} \), and \( \text{navigationParams}' \text{ request} \).

12. If \( \text{navigationParams}' \text{ request} \) is non-null, then:
   1. Set \( \text{document}' \text{ referrer} \) to the empty string.
   2. Let \( \text{referrer} \) be \( \text{navigationParams}' \text{ request}' \text{ referrer} \).
   3. If \( \text{referrer} \) is a \text{URL record}, then set \( \text{document}' \text{ referrer} \) to the serialization of \( \text{referrer} \).

   **Note**
   Per Fetch, \( \text{referrer} \) will be either a \text{URL record} or "no-referrer" at this point.

13. If \( \text{navigationParams}' \text{ response} \) has a \`Refresh\' header, then:
   1. Let \( \text{value} \) be the isomorphic decoding of the value of the header.
   2. Run the \text{shared declarative refresh steps} with \( \text{document} \) and \( \text{value} \).

   We do not currently have a spec for how to handle multiple \`Refresh\' headers. This is tracked as issue #2900.

14. Return \( \text{document} \).

---

**Example**

In this example, the child document is not allowed to use \( \text{PaymentRequest} \), despite being same origin-domain at the time the child document tries to use it. At the time the child document is initialized, only the parent document has set \( \text{document}.'\text{domain} \), and the child document has not.

```html
<!doctype html>
<script>
  document.domain = 'example.com';
</script>
<iframe src='b.html'></iframe>

<!doctype html>
```

883
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, given some navigation params `navigationParams` and a `Document` `newDocument`. When a user agent is required to do this, it must queue a global task on the networking task source, given the relevant global object of the `Document` object of the current entry (not the new one), to run the following steps:

1. **Unload** the `document` of the current entry.
   
   If this instance of the navigation 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 navigation 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 `navigationParams`'s history handling is "entry update" or "reload"
   
   1. Replace the `document` of the current entry, and of any other entries that reference the same document, as that entry, with newDocument.
   2. Traverse the history to the current entry with historyHandling set to navigationParams's history handling.

   Otherwise, if the navigation was initiated with a URL that equals navigationParams's browsing context's active document's URL
      
   1. Let `newEntry` be a new session history entry whose URL is newDocument's URL and document is newDocument.

      ```
      Some browsers copy over the serialized state of current entry, but this is inconsistent. See issue #6213 for more discussion on this.
      ```
   2. Insert `newEntry` after the current entry in navigationParams's browsing context's session history.
   3. Traverse the history to `newEntry` with historyHandling set to "replace".

   Otherwise, if navigationParams's history handling is "replace"
      
   1. Let `newEntry` be a new session history entry whose URL is newDocument's URL and document is newDocument.
2. Insert newEntry after the current entry in navigationParams's browsing context's session history.

3. Traverse the history to newEntry with historyHandling set to "replace".

Otherwise

1. Assert: navigationParams's history handling is "default".

2. Remove all the entries in the 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.

3. Append a new session history entry to the session history, whose URL is newDocument's URL and document is newDocument.

4. Traverse the history to the new entry.

3. The navigation algorithm has now matured.

4. Try to scroll to the fragment for newDocument.

To try to scroll to the fragment for a Document document, perform the following steps in parallel:

1. Wait for an implementation-defined amount of time. (This is intended to allow the user agent to optimize the user experience in the face of performance concerns.)

2. Queue a global task on the networking task source given document's relevant global object to run these steps:
   1. If document 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, then abort these steps.
   2. Scroll to the fragment given in document's URL. If this does not find an indicated part of the document, then try to scroll to the fragment for document.

7.10.2 Page load processing model for HTML files

When an HTML document is to be loaded, given navigation params navigationParams, the user agent must queue a task on the networking task source to:

1. Let document be the result of creating and initializing a Document object given "html", "text/html", and navigationParams.

2. 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 global task on the networking task source given the newly-created Document's relevant global object 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 given navigationParams and the newly-created Document.
7.10.3 Page load processing model for XML files

When faced with displaying an XML file inline, provided navigationParams and a string type, user agents must follow the requirements defined in XML and Namespaces in XML, XML Media Types, DOM, and other relevant specifications to create and initialize a Document object given "xml", type, and navigationParams. They must also create a corresponding XML parser given navigationParams and a string type.

Note
At the time of writing, the XML specification community had not actually yet specified how XML and the DOM interact.

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.

Then, with the newly created Document, the user agent must update the session history with the new page given navigationParams and the newly-created Document. 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.

7.10.4 Page load processing model for text files

When a plain text document is to be loaded, provided navigationParams and a string type, the user agent must queue a task on the networking task source to:

1. Let document be the result of creating and initializing a Document object given "html", type, and navigationParams.
2. Create an HTML parser and 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 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 (i.e., type).

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 global task on the networking task source given the newly-created Document's relevant global object 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 given navigationParams and the newly-created Document.

User agents may add content to the head element of the Document, e.g., linking to a style sheet, providing script, or giving the document a title.

Note
In particular, if the user agent supports the Format=Flowed feature of RFC 3676 then the user agent would need to apply extra styling to cause the text to wrap correctly and to handle the quoting feature. This could be performed using, e.g., a CSS extension.

7.10.5 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.
This algorithm is passed \texttt{navigation params}, but it’s unclear how exactly to use them.

For each body part obtained from the resource, the user agent must run \texttt{process a navigate response} using the new body part and the same \texttt{browsing context}, with \texttt{history handling} set to “\texttt{replace}” if a previous body part from the same resource resulted in a creating and initializing a \texttt{Document object}, and otherwise using the same setup as the \texttt{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.

\textbf{Note}\hfill\textit{Thus, load events (and for that matter unload events) do fire for each body part loaded.}

7.10.6 Page load processing model for media

When an image, video, or audio resource is to be loaded, provided \texttt{navigation params} \texttt{navigationParams} and a string \texttt{type}, the user agent should:

1. Let \texttt{document} be the result of \texttt{creating and initializing a Document object} given “\texttt{html}”, type, and \texttt{navigationParams}.
2. Append an \texttt{html} element to \texttt{document}.
3. Append a \texttt{head} element to the \texttt{html} element.
4. Append a \texttt{body} element to the \texttt{html} element.
5. Append an element \texttt{host element} for the media, as described below, to the \texttt{body} element.
6. Set the appropriate attribute of the element \texttt{host element}, as described below, to the address of the image, video, or audio resource.

The element \texttt{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.

<table>
<thead>
<tr>
<th>Type of media</th>
<th>Element for the media</th>
<th>Appropriate attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>\texttt{img}</td>
<td>\texttt{src}</td>
</tr>
<tr>
<td>Video</td>
<td>\texttt{video}</td>
<td>\texttt{src}</td>
</tr>
<tr>
<td>Audio</td>
<td>\texttt{audio}</td>
<td>\texttt{src}</td>
</tr>
</tbody>
</table>

Then, the user agent must act as if it had \texttt{stopped parsing}.

After creating the \texttt{Document} object, but potentially before the page has finished fully loading, the user agent must update the session history with the new page given \texttt{navigationParams} and the newly-created \texttt{Document}.

User agents may add content to the \texttt{head} element of the \texttt{Document}, or attributes to the element \texttt{host element}, e.g., to link to a style sheet, to provide a script, to give the document a \texttt{title}, or to make the media \texttt{autoplay}.

7.10.7 Page load processing model for content that uses plugins

When a resource that requires an external resource to be rendered is to be loaded, provided \texttt{navigation params} \texttt{navigationParams} and a string \texttt{type}, the user agent should:

1. Let \texttt{document} be the result of \texttt{creating and initializing a Document object} given “\texttt{html}”, type, and \texttt{navigationParams}.
2. Mark \texttt{document} as being a \texttt{plugin document}.
3. Append an \texttt{html} element to \texttt{document}.
4. Append a \texttt{head} element to the \texttt{html} element.
5. Append a `body` element to the `html` element.
6. Append an `embed` to the `body` element.
7. 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 `iframe` can't be used to evade plugin-types directives. [CSP]

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 given `navigationParams` and the newly-created `Document`.

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 or to give the document a `title`.

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.

7.10.8 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, provided a `browsing context` `browsingContext`, the user agent should:

1. Let `navigationParams` be a new `navigation params` whose `request` is null, `response` is null, `origin` is a new opaque origin, final sandboxing flag set is an empty set, `cross-origin opener policy` is a new `cross-origin opener policy`, `COOP enforcement result` is a new `cross-origin opener policy enforcement result`, `reserved environment` is null, and `browsing context` is `browsingContext`.

   The algorithm called in the next step is not prepared to deal with a null `response`. Probably we should synthesize one instead.

2. Let `document` be the result of creating and initializing a `Document object` given "html", "text/html", and `navigationParams`.

3. Either associate `document` with a custom rendering that is not rendered using the normal `Document` rendering rules, or mutate `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 given `navigationParams` and the newly-created `Document`.

7.10.9 Navigating to a fragment

To navigate to a fragment given a `browsing context` `browsingContext`, a `URL` `url`, and a `history handling behavior` `historyHandling`:

1. If `historyHandling` is not "replace", then remove all the entries in `browsingContext`'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 `browsingContext`'s top-level browsing context's document family.
3. Append a new `session history entry` to the `session history` whose `URL` is `url`, `document` is the `current entry`'s `document`, and `scroll restoration mode` is the `current entry`'s `scroll restoration mode`.

4. Traverse the `history` to the new entry, with `historyHanding` set to `historyHanding` and with `nonBlockingEvents` set to true. This will `scroll to the fragment` given in what is now the document's `URL`.

**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` 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. If there is no `indicated part of the document`, set the `Document`'s `target element` to null.
2. If the `indicated part of the document` is the top of the document, then:
   1. Set the `Document`'s `target element` to null.
   2. `Scroll to the beginning of the document` for the `Document`. [CSSOMVIEW]
3. Otherwise:
   1. Let `target` be element that is the `indicated part of the document`.
   2. Set the `Document`'s `target element` to `target`.
   3. `Scroll target into view`, with `behavior` set to "auto", `block` set to "start", and `inline` set to "nearest". [CSSOMVIEW]
   4. Run the `focusing steps` for `target`, 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`, if any, identifies. The semantics of the `fragment` in terms of mapping it to a node is defined by the specification that defines the `MIME type` used by the `Document` (for example, the processing of `fragments` for `XML MIME types` is the responsibility of RFC7303). [RFC7303]

There is also a `target element` for each `Document`, which is used in defining the `:target` pseudo-class and is updated by the above algorithm. It is initially null.

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. Let `fragment` be the document's `URL`'s `fragment`.
2. If `fragment` is the empty string, then the `indicated part of the document` is the top of the document; return.
3. If `find a potential indicated element` with `fragment` returns non-null, then the return value is the `indicated part of the document`; return.
4. Let `decodedBytes` be the result of `percent-decoding` `fragment`.
5. Let `decodedFragment` be the result of running `UTF-8 decode without BOM` on `decodedBytes`.
6. If `find a potential indicated element` with `decodedFragment` returns non-null, then the return value is the `indicated part of the document`; return.
7. If `decodedFragment` is an `ASCII case-insensitive` match for the string `top`, then the `indicated part of the document` is the top of the document; return.
8. There is no `indicated part of the document`.

To `find a potential indicated element` given a string `fragment`, run these steps:
1. If there is an element in the document tree that has an ID equal to fragment, then return the first such element in tree order.
2. If there is an element in the document tree that has a name attribute whose value is equal to fragment, then return the first such element in tree order.
3. Return null.

7.10.10 History traversal

To traverse the history to a session history entry entry, with an optional history handling behavior historyHandling (default "default"), an optional boolean nonBlockingEvents (default false), and an optional boolean explicitHistoryNavigation (default false):

1. If entry's document is null, then:
   1. Assert: historyHandling is "default".
   2. Let request be a new request whose url is entry's URL.
   3. If explicitHistoryNavigation is true, then set request's history-navigation flag.
   4. Navigate the browsing context to request with historyHandling set to "entry update". The navigation must be done using the same source browsing context as was used the first time entry was created.
5. Return.

2. If the current entry's title is null, then set its title to the value returned by the document.title IDL attribute.
3. Save persisted state to the current entry.
4. Let newDocument be entry's document.
5. If newDocument is different than the current entry's document, or historyHandling is "entry update" or "reload", then:
   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 newDocument's origin is not same origin with the current entry's document's origin, then:
      1. Let entriesToUpdate be all entries in the session history whose document's origin is same origin as the active document and that are contiguous with the current entry.
      2. For each entryToUpdate of entriesToUpdate, set entryToUpdate's browsing context name to the current browsing context name.
      3. If the browsing context is a top-level browsing context, but not an auxiliary browsing context whose disowned is false, then set the browsing context's name to the empty string.
   3. Set the active document of the browsing context to newDocument.

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.

Note

The "navigate" algorithm reinvokes this "traverse" algorithm to complete the traversal, at which point entry's document is non-null.

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.
4. If entry's browsing context name is not null, then:

1. Set the browsing context's name to entry's browsing context name.

2. Let entriesToUpdate be all entries in the session history whose document's origin is same origin as the new active document's origin and that are contiguous with entry.

3. For each entryToUpdate of entriesToUpdate, set entryToUpdate's browsing context name to null.

5. If newDocument has any form controls whose autofill field name is "off", invoke the reset algorithm of each of those elements.

6. If newDocument's current document readiness "complete", then queue a global task on the DOM manipulation task source given newDocument's relevant global object to run the following steps:

1. If newDocument's page showing flag is true, then abort these steps.

2. Set newDocument's page showing flag to true.

3. Run any session history document visibility change steps for newDocument that are defined by other applicable specifications.

Note: This is specifically intended for use by Page Visibility. [PAGEVIS]

4. Fire an event named pageshow at newDocument's relevant global object, using PageTransitionEvent, with the persisted attribute initialized to true, and legacy target override flag set.

6. Set newDocument's URL to entry's URL.

7. Let hashChanged be false, and let oldURL and newURL be null.

8. If entry's URL is fragment not identical to the current entry's URL's fragment, and entry's document equals the current entry's document, then set hashChanged to true, set oldURL to the current entry's URL, and set newURL to entry's URL.

9. If historyHandling is "replace", then remove the entry immediately before entry in the session history.

10. If entry's persisted user state is null, and its URL's fragment is non-null, then scroll to the fragment.

11. Set the current entry to entry.

12. Let targetRealm be the current Realm Record.

13. Let state be null.

14. If entry's serialized state is not null, then set state to StructuredDeserialize(entry's serialized state, targetRealm). If this throws an exception, catch it and ignore the exception.

15. Set newDocument's History object's state to state.

16. Let stateChanged be true if newDocument has a latest entry, and that entry is not entry; otherwise let it be false.

17. Set newDocument's latest entry to entry.

18. If nonBlockingEvents is false, then run the following steps immediately. Otherwise, queue a global task on the DOM manipulation task source given newDocument's relevant global object to run the following steps instead.

1. If stateChanged is true, then fire an event named popstate at newDocument's relevant global object, using PopStateEvent, with the state attribute initialized to state.

2. Restore persisted state from entry.

3. If hashChanged is true, then fire an event named hashchange at the browsing context's active window, using HashChangeEvent, with the oldURL attribute initialized to oldURL and the newURL attribute initialized to newURL.
To save persisted state to a session history entry:

1. Set the scroll position data of entry to contain the scroll positions for all of entry's document's restorable scrollable regions.
2. Optionally, update entry's persisted user state to reflect any state that the user agent wishes to persist, such as the values of form fields.

To restore persisted state from a session history entry:

1. If entry's scroll restoration mode is "auto", then the user agent may use entry's scroll position data to restore the scroll positions of entry's document's restorable scrollable regions.
   
   Note: The user agent not restoring scroll positions does not imply that scroll positions will be left at any particular value (e.g., (0,0)). The actual scroll position depends on the navigation type and the user agent's particular caching strategy. So web applications cannot assume any particular scroll position but rather are urged to set it to what they want it to be.

2. Optionally, update other aspects of entry's document and its rendering, for instance values of form fields, that the user agent had previously recorded in entry's persisted user state.

   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.

The restorable scrollable regions of a document document are document's viewport, and all of document's scrollable regions excepting any child browsing contexts of document.

Note: Child browsing context scroll restoration is handled by the history entry for those browsing contexts' Document's.

7.10.10.2 The PopStateEvent interface

IDL [Exposed=Window]

interface PopStateEvent : Event {
    constructor(DOMString type, optional PopStateEventInit eventInitDict = {});

    readonly attribute any state;
};

dictionary PopStateEventInit : EventInit {
    any state = null;
};

For web developers (non-normative)

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. It represents the context information for the event, or null, if the state represented is the initial state of the Document.
7.10.10.3 The `HashChangeEvent` interface

IDL

```idl
[Exposed=Window]
interface HashChangeEvent : Event {
    constructor(DOMString type, optional HashChangeEventInit eventInitDict = {});

    readonly attribute USVString oldURL;
    readonly attribute USVString newURL;
};
```

dictionary `HashChangeEventInit` : EventInit {
    USVString oldURL = "";
    USVString newURL = "";
};

For web developers (non-normative)

- **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. 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. It represents context information for the event, specifically the URL of the session history entry that was traversed to.

7.10.10.4 The `PageTransitionEvent` interface

IDL

```idl
[Exposed=Window]
interface PageTransitionEvent : Event {
    constructor(DOMString type, optional PageTransitionEventInit eventInitDict = {});

    readonly attribute boolean persisted;
};
```

dictionary `PageTransitionEventInit` : EventInit {
    boolean persisted = false;
};

For web developers (non-normative)

- **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:
  - The user agent decided to not keep the `Document` alive in a session history entry after `unload`
  - Having `iframe`s that are not salvageable
  - Active `WebSocket` objects
  - `Abort` a `Document`

The `persisted` attribute must return the value it was initialized to. It represents the context information for the event.
# 7.10.11 Loading documents

A Document has a completely loaded time (a time or null), which is initially null.

A Document is considered completely loaded if its completely loaded time is non-null.

To completely finish loading a Document document:

1. Assert: document's browsing context is non-null.
2. Set document's completely loaded time to the current time.
3. Let container be document's browsing context's container.
4. If container is an iframe element, then queue an element task on the DOM manipulation task source given container to run the iframe load event steps given container.
5. Otherwise, if container is non-null, then queue an element task on the DOM manipulation task source given container to fire an event named load at container.

# 7.10.12 Unloading documents

A Document has a salvageable state, which must initially be true, 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 0.

To prompt to unload, given a Document object document and optionally a recursiveFlag, run these steps:

1. Increase the event loop's termination nesting level by 1.
2. Increase the document's ignore-opens-during-unload counter by 1.
3. Let event be the result of creating an event using BeforeUnloadEvent.
4. Initialize event's type attribute to beforeunload and its cancelable attribute true.
5. Dispatch: Dispatch event at document's relevant global object.
6. Decrease the event loop's termination nesting level by 1.
7. If all of the following are true:
   - document's active sandboxing flag set does not have its sandboxed modals flag set
   - document's relevant global object has sticky activation
   - event's canceled flag is set, or the returnValue attribute of event is not the empty string

then the user agent may ask the user to confirm that they wish to unload the document.

Note

The message shown to the user is not customizable, but instead determined by the user agent. In particular, the actual value of the returnValue attribute is ignored.

The user agent is encouraged to avoid asking the user for confirmation if it judges that doing so would be annoying, deceptive, or pointless.

If the user agent asks the user for confirmation, it 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 the recursiveFlag is not set, then:
   1. Let descendants be the list of the descendant browsing contexts of document.
   2. For each browsingContext in descendants:
1. **Prompt to unload** browsingContext's active document with the recursiveFlag set.

2. If the user refused to allow the document to be unloaded, then the user implicitly also refused to allow document to be unloaded; break.


To **unload** a Document document, optionally given a recursiveFlag:

1. Increase the event loop's termination nesting level by one.

2. Increase document's ignore-opens-during-unload counter by one.

3. If 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 document's page showing flag to false.

5. If the user agent does not intend to keep document alive in a session history entry (such that it can be reused later on history traversal), set document's salvageable state to false.

6. Fire an event named pagehide at document's relevant global object, using PageTransitionEvent, with the persisted attribute initialized to true if document's salvageable state is true, and false otherwise, and legacy target override flag set.

7. Run any unloading document visibility change steps for document that are defined by other applicable specifications.

**Note**

This is specifically intended for use by Page Visibility. [PAGEVIS]

8. Unload event: If document's salvageable state is false, then fire an event named unload at document's relevant global object, with legacy target override flag set.

9. Decrease the event loop's termination nesting level by one.

10. Run any unloading document cleanup steps for document that are defined by this specification and other applicable specifications.

11. If the recursiveFlag is not set, then:

   1. Let descendants be the list of the descendant browsing contexts of document.

   2. For each browsingContext in descendants:

      1. Unload the active document of browsingContext with the recursiveFlag set.

      2. If the salvageable state of the active document of browsingContext is false, then set the salvageable state of document to false also.

      3. If document's salvageable state is false, then discard document.


This specification defines the following **unloading document cleanup steps**. Other specifications can define more. Given a Document document:

1. Let window be document's relevant global object.

2. For each WebSocket object webSocket whose relevant global object is window, make disappear webSocket. If this affected any WebSocket objects, then set document's salvageable state to false.

3. If document's salvageable state is false, then:

   1. For each EventSource object eventSource whose relevant global object is equal to window, forcibly close eventSource.

   2. Empty window's list of active timers.
### 7.10.12.1 The `BeforeUnloadEvent` interface

```idl
[Exposed=Window]
interface BeforeUnloadEvent : Event {
    attribute DOMString returnValue;
};
```

Note

There are no `BeforeUnloadEvent`-specific initialization methods.

The `BeforeUnloadEvent` interface is a legacy interface which allows prompting to unload to be controlled not only by canceling the event, but by setting the `returnValue` attribute to a value besides the empty string. Authors should use the `preventDefault()` method, or other means of canceling events, instead of using `returnValue`.

The `returnValue` attribute controls the process of prompting to unload. 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.

Note

This attribute is a DOMString only for historical reasons. Any value besides the empty string will be treated as a request to ask the user for confirmation.

### 7.10.13 Aborting a document load

To abort a `Document` document:

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 document's `salvageable` state to false also.
2. Cancel any instances of the `fetch` algorithm in the context of `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 `document`'s `salvageable` state to false.
3. If `document` has an active parser, then:
   1. Set document's `active parser was aborted` to true.
   2. Abort that parser.
   3. Set `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 an event named `abort` at that `Document` object's relevant global object before invoking the `abort` algorithm.

To stop document loading given a `Document` object `document`, run these steps:

1. If `document` is not an active document, then return.
2. Let `browsingContext be document's browsing context`.
3. If there is an existing attempt to navigate `browsingContext` and that attempt is not currently running the `unload a document` algorithm, then cancel that `navigation`.

### 7.10.14 The `X-Frame-Options` header

The `X-Frame-Options` HTTP response header is a legacy way of controlling whether and how a `Document` may be loaded.
inside of a child browsing context\(^{816}\). It is obsoleted by the `frame-ancestors` CSP directive, which provides more granular control over the same situations. It was originally defined in HTTP Header Field X-Frame-Options, but the definition and processing model here supersedes that document. [CSP]\(^{9170}\) [RFC7034]\(^{1276}\).

**Note**

In particular, HTTP Header Field X-Frame-Options specified an `ALLOW-FROM` variant of the header, but that is not to be implemented.

**Note**

Per the below processing model, if both a CSP `frame-ancestors` directive and an `X-Frame-Options`\(^{1245}\) header are used in the same response, then `X-Frame-Options`\(^{1245}\) is ignored.

For web developers and conformance checkers, its value ABNF is:

\[
\text{X-Frame-Options} = \text{"DENY" / "SAMEORIGIN"}
\]

To check a navigation response’s adherence to `X-Frame-Options`, given a response response, a browsing context\(^{813}\) browsingContext, and an origin\(^{840}\) destinationOrigin:

1. If `response`'s CSP list
2. **For each** policy in `response`'s CSP list:
   1. If policy's disposition is not "enforce", then **continue**.
   2. If policy's directive set contains a `frame-ancestors` directive, then return true.
3. Let `rawXFrameOptions` be the result of getting, decoding, and splitting `X-Frame-Options`\(^{1245}\) from `response`'s header list.
4. Let `xFrameOptions` be a new set.
5. **For each** value of `rawXFrameOptions`, append value, converted to ASCII lowercase, to `xFrameOptions`.
6. If `xFrameOptions`'s `size` is greater than 1, and `xFrameOptions` contains any of "deny", "allowall", or "sameorigin", then return false.

**Note**

The intention here is to block any attempts at applying `X-Frame-Options`\(^{1245}\), which were trying to do something valid, but appear confused.

**Note**

This is the only impact of the legacy `ALLOWALL` value on the processing model.

7. If `xFrameOptions`'s `size` is greater than 1, then return true.

**Note**

This means it contains multiple invalid values, which we treat the same way as if the header was omitted entirely.

8. If `xFrameOptions[0]` is "deny", then return false.
9. If `xFrameOptions[0]` is "sameorigin", then:
   1. Let `containerDocument` be `browsingContext`'s container document\(^{816}\).
   2. **While** `containerDocument` is not null:
      1. If `containerDocument`'s origin is not same origin\(^{841}\) with `destinationOrigin`, then return false.
      2. Let `containerBC` be `containerDocument`'s browsing context\(^{813}\).
      3. Set `containerDocument` to `containerBC`'s container document\(^{816}\), if `containerBC` is non-null; otherwise, null.
10. Return true.
If we've reached this point then we have a lone invalid value (which could potentially be one the legacy `ALLOWALL` or `ALLOW-FROM` forms). These are treated as if the header were omitted entirely.

### Example

The following table illustrates the processing of various values for the header, including non-conformant ones:

<table>
<thead>
<tr>
<th><code>X-Frame-Options</code></th>
<th>Valid</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>'DENY'</td>
<td>✓</td>
<td>embedding disallowed</td>
</tr>
<tr>
<td>'SAMEORIGIN'</td>
<td>✓</td>
<td>same-origin embedding allowed</td>
</tr>
<tr>
<td>'INVALID'</td>
<td>x</td>
<td>embedding allowed</td>
</tr>
<tr>
<td>'ALLOWALL'</td>
<td>x</td>
<td>embedding allowed</td>
</tr>
<tr>
<td>'ALLOW-FROM=<a href="https://example.com/">https://example.com/</a>'</td>
<td>x</td>
<td>embedding allowed (from anywhere)</td>
</tr>
</tbody>
</table>

### Example

The following table illustrates how various non-conformant cases involving multiple values are processed:

<table>
<thead>
<tr>
<th><code>X-Frame-Options</code></th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>'SAMEORIGIN, SAMEORIGIN'</td>
<td>same-origin embedding allowed</td>
</tr>
<tr>
<td>'SAMEORIGIN, DENY'</td>
<td>embedding disallowed</td>
</tr>
<tr>
<td>'SAMEORIGIN,'</td>
<td>embedding disallowed</td>
</tr>
<tr>
<td>'SAMEORIGIN, ALLOWALL'</td>
<td>embedding disallowed</td>
</tr>
<tr>
<td>'SAMEORIGIN, INVALID'</td>
<td>embedding disallowed</td>
</tr>
<tr>
<td>ALLOWALL, INVALID'</td>
<td>embedding disallowed</td>
</tr>
<tr>
<td>ALLOWALL,'</td>
<td>embedding disallowed</td>
</tr>
<tr>
<td>INVALID, INVALID'</td>
<td>embedding allowed</td>
</tr>
</tbody>
</table>

The same results are obtained whether the values are delivered in a single header whose value is comma-delimited, or in multiple headers.
8 Web application APIs

8.1 Scripting

8.1.1 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.

8.1.2 Agents and agent clusters

8.1.2.1 Integration with the JavaScript agent formalism

JavaScript defines the concept of an agent. This section gives the mapping of that language-level concept on to the web platform.

---

**Note**

Conceptually, the agent concept is an architecture-independent, idealized "thread" in which JavaScript code runs. Such code can involve multiple globals/realms that can synchronously access each other, and thus needs to run in a single execution thread.

Two `Window` objects having the same agent does not indicate they can directly access all objects created in each other's realms. They would have to be same origin-domain; see `IsPlatformObjectSameOrigin`.

The following types of agents exist on the web platform:

**Similar-origin window agent**

Contains various `Window` objects which can potentially reach each other, either directly or by using `document.domain`.

If the encompassing agent cluster's is origin-keyed is true, then all the `Window` objects will be same origin, can reach each other directly, and `document.domain` will no-op.

**Note**

Two `Window` objects that are same origin can be in different similar-origin window agents, for instance if they are each in their own browsing context group.

**Dedicated worker agent**

Contains a single `DedicatedWorkerGlobalScope`.

**Shared worker agent**

Contains a single `SharedWorkerGlobalScope`.

**Service worker agent**

Contains a single `ServiceWorkerGlobalScope`.

**Worklet agent**

Contains a single `WorkletGlobalScope` object.
Only shared\textsuperscript{989} and dedicated worker agents\textsuperscript{989} allow the use of JavaScript Atomics APIs to potentially block.

To create an agent, given a boolean canBlock:

1. Let signifier be a new unique internal value.
2. Let candidateExecution be a new candidate execution.
3. Let agent be a new agent whose \([\text{CanBlock}]\) is canBlock, \([\text{Signifier}]\) is signifier, \([\text{CandidateExecution}]\) is candidateExecution, and \([\text{IsLockFree1}], [\text{IsLockFree2}], \text{and}[\text{LittleEndian}]\) are set at the implementation's discretion.
4. Set agent's event loop\textsuperscript{931} to a new event loop\textsuperscript{931}.
5. Return agent.

The relevant agent for a platform object platformObject is platformObject's relevant Realm\textsuperscript{910}'s agent. This pointer is not yet defined in the JavaScript specification; see tc39/ecma262\#1357.

An agent cluster has an associated cross-origin isolation mode, which is a cross-origin isolation mode\textsuperscript{921}. It is initially "none\textsuperscript{921}".

An agent cluster has an associated is origin-keyed (a boolean), which is initially false.

The following defines the allocation of the agent clusters of similar-origin window agents\textsuperscript{989}.

An agent cluster key is a site\textsuperscript{841} or tuple origin\textsuperscript{840}. Without web developer action to achieve origin-keyed agent clusters\textsuperscript{843}, it will be a site\textsuperscript{841}.

An equivalent formulation is that an agent cluster key\textsuperscript{900} can be a scheme-and-host\textsuperscript{841} or an origin\textsuperscript{860}.

To obtain a similar-origin window agent, given an origin\textsuperscript{840} origin, a browsing context group\textsuperscript{9820} group, and a boolean requestsOAC, run these steps:

1. Let site be the result of obtaining a site\textsuperscript{841} with origin.
2. Let key be site.
3. If group's cross-origin isolation mode\textsuperscript{821} is not "none\textsuperscript{821}", then set key to origin.
4. Otherwise, if group's historical agent cluster key map\[origin] exists, then set key to group's historical agent cluster key map\[origin].

5. Otherwise:
   1. If requestsOAC is true, then set key to origin.
   2. Set group's historical agent cluster key map\[origin] to key.

6. If group's agent cluster map\[key] does not exist, then:
   1. Let agentCluster be a new agent cluster.
   2. Set agentCluster's cross-origin isolation mode to group's cross-origin isolation mode.
   3. Set agentCluster's is origin-keyed to true if key equals origin; otherwise false.
   4. Add the result of creating an agent, given false, to agentCluster.
   5. Set group's agent cluster map\[key] to agentCluster.

7. Return the single similar-origin window agent contained in group's agent cluster map\[key].

Note
This means that there is only one similar-origin window agent per browsing context agent cluster. (However, dedicated worker and worklet agents might be in the same cluster.)

The following defines the allocation of the agent clusters of all other types of agents.

To obtain a worker/worklet agent, given an environment settings object or null outside settings, a boolean isTopLevel, and a boolean canBlock, run these steps:

1. Let agentCluster be null.
2. If isTopLevel is true, then:
   1. Set agentCluster to a new agent cluster.
   2. Set agentCluster's is origin-keyed to true.

Note
These workers can be considered to be origin-keyed. However, this is not exposed through any APIs (in the way that originAgentCluster exposes the origin-keyedness for windows).

3. Otherwise:
   1. Assert: outside settings is not null.
   2. Let ownerAgent be outside settings's Realm's agent.
   3. Set agentCluster to the agent cluster which contains ownerAgent.
4. Let agent be the result of creating an agent given canBlock.
5. Add agent to agentCluster.
6. Return agent.

To obtain a dedicated/shared worker agent, given an environment settings object or null outside settings and a boolean isShared, return the result of obtaining a worker/worklet agent given outside settings, isShared, and true.

To obtain a worklet agent, given an environment settings object or null outside settings, return the result of obtaining a worker/worklet agent given outside settings, false, and false.

To obtain a service worker agent, return the result of obtaining a worker/worklet agent given null, true, and false.
The JavaScript specification introduces the **realm** concept, representing a global environment in which script is run. Each realm comes with an implementation-defined **global object**; much of this specification is devoted to defining that global object and its properties.

For web specifications, it is often useful to associate values or algorithms with a realm/global object pair. When the values are specific to a particular type of realm, they are associated directly with the global object in question, e.g., in the definition of the **Window** or **WorkerGlobalScope** interfaces. When the values have utility across multiple realms, we use the **environment settings object** concept.

Finally, in some cases it is necessary to track associated values before a realm/global object/environment settings object even comes into existence (for example, during **navigation**). These values are tracked in the **environment** concept.

### 8.1.3 Realms and their counterparts

The JavaScript specification introduces the **realm** concept, representing a global environment in which script is run. Each realm comes with an implementation-defined **global object**; much of this specification is devoted to defining that global object and its properties.

For web specifications, it is often useful to associate values or algorithms with a realm/global object pair. When the values are specific to a particular type of realm, they are associated directly with the global object in question, e.g., in the definition of the **Window** or **WorkerGlobalScope** interfaces. When the values have utility across multiple realms, we use the **environment settings object** concept.

Finally, in some cases it is necessary to track associated values before a realm/global object/environment settings object even comes into existence (for example, during **navigation**). These values are tracked in the **environment** concept.

#### 8.1.3.1 Environments

An **environment** is an object that identifies the settings of a current or potential execution environment. An **environment** has the following fields:

**An id**

An opaque string that uniquely identifies this **environment**.

**A creation URL**

A **URL** that represents the location of the resource with which this **environment** is associated.

**Note**

In the case of an **environment settings object**, this **URL** might be distinct from the **environment settings object**’s responsible document’s **URL**, due to mechanisms such as **history.pushState()**.

**A top-level creation URL**

Null or a **URL** that represents the **creation URL** of the “top-level” **environment**. It is null for workers and worklets.

**A top-level origin**

A for now implementation-defined value, null, or an **origin**. For a “top-level” potential execution environment it is null (i.e., when there is no response yet); otherwise it is the “top-level” **environment**’s **origin**. For a dedicated worker or worklet it is the **top-level origin** of its creator. For a shared or service worker it is an implementation-defined value.

**Note**

This is distinct from the **top-level creation URL**’s **origin** when sandboxing, workers, and worklets are involved.
A target browsing context
Null or a target browsing context for a navigation request.

An active service worker
Null or a service worker that controls the environment.

An execution ready flag
A flag that indicates whether the environment setup is done. It is initially unset.

Specifications may define environment discarding steps for environments. The steps take an environment as input.

Note
The environment discarding steps are run for only a select few environments: the ones that will never become execution ready because, for example, they failed to load.

8.1.3.2 Environment settings objects

An environment settings object is an environment that additionally specifies algorithms for:

A realm execution context
A JavaScript execution context shared by all scripts that use this settings object, i.e. all scripts in a given JavaScript realm. When we run a classic script or run a module 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 and Source Text Module Record's Evaluate know which Realm to use.)

A module map
A module map that is used when importing JavaScript modules.

A responsible document
A Document that is assigned responsibility for actions taken by the scripts that use this environment settings object.

Example
For example, the URL of the responsible document is used to set the URL of the Document after it has been reset using document.open().

If the responsible event loop is not a window 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
A URL used by APIs called by scripts that use this environment settings object to parse URLs.

An origin
An origin used in security checks.

A referrer policy
The default referrer policy for fetches performed using this environment settings object as a request client.

An embedder policy
An embedder policy used by cross-origin resource policy checks for fetches performed using this environment settings object as a request client.

A cross-origin isolated capability
A boolean representing whether scripts that use this environment settings object are allowed to use APIs that require cross-origin isolation.
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.

An environment settings object's responsible event loop is its global object's relevant agent's event loop.

### 8.1.3.3 Realms, settings objects, and global objects

A global object is a JavaScript object that is the [[GlobalObject]] field of a JavaScript realm.

> In this specification, all JavaScript realms are created with global objects that are either Window or WorkerGlobalScope objects.

There is always a 1-to-1-to-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 creation 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.

To create a new JavaScript realm in an agent, optionally with instructions to create a global object or a global this binding (or both), the following steps are taken:

1. Perform InitializeHostDefinedRealm() with the provided customizations for creating the global object and the global this binding.
2. Let realm execution context be the running JavaScript execution context.

> This is the JavaScript execution context created in the previous step.

3. Remove realm execution context from the JavaScript execution context stack.
4. Let realm be realm execution context's Realm component.
5. Set realm's agent to agent. This pointer is not yet defined in the JavaScript specification; see tc39/ecma262#1357.
6. If agent's agent cluster's cross-origin isolation mode is "none", then:
   1. Let global be realm's global object.
   2. Let status be ! global.[[Delete]]("SharedArrayBuffer").
   3. Assert: status is true.

> This is done for compatibility with web content and there is some hope that this can be removed in the future. Web developers can still get at the constructor through new WebAssembly.Memory({ shared:true, initial:0, maximum:0 }).buffer.constructor.

7. Return realm execution context.
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, or the author function or script that originally scheduled the currently-running callback.

**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, which is roughly the JavaScript realm in which it was created. 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.

⚠ **Warning!**
The incumbent and entry concepts should not be used by new specifications, as they are excessively complicated and unintuitive to work with. We are working to remove almost all existing uses from the platform: see issue #1430 for incumbent, and issue #1431 for entry.

In general, web platform specifications should use the relevant concept, applied to the object being operated on (usually the this value of the current method). This mismatches the JavaScript specification, where current is generally used as the default (e.g. in determining the JavaScript realm whose Array constructor should be used to construct the result in Array.prototype.map). But this inconsistency is so embedded in the platform that we have to accept it going forward.

⚠ **Warning!**
The incumbent and entry concepts should not be used by new specifications, as they are excessively complicated and unintuitive to work with. We are working to remove almost all existing uses from the platform: see issue #1430 for incumbent, and issue #1431 for entry.

---

Example
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):

```html
<!-- 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`.

Example

One reason why the relevant concept is generally a better default choice than the current concept is that it is more suitable for creating an object that is to be persisted and returned multiple times. For example, the `navigator.getBattery()` method creates promises in the relevant Realm for the `Navigator` object on which it is invoked. This has the following impact:

```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>
</html>
```

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.
8.1.3.3.1 Entry  

The process of calling scripts will push or pop realm execution contexts onto the JavaScript execution context stack, interspersed with other execution contexts.

With this in hand, we define the entry execution context to be the most recently pushed item in the JavaScript execution context stack that is a realm execution context. 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.

8.1.3.3.2 Incumbent  

All JavaScript execution contexts must contain, as part of their code evaluation state, a skip-when-determining-incumbent counter value, which is initially zero. In the process of preparing to run a callback and cleaning up after running a callback, this value will be incremented and decremented.

Every event loop has an associated backup incumbent settings object stack, initially empty. Roughly speaking, it is used to determine the incumbent settings object when no author code is on the stack, but author code is responsible for the current algorithm having been run in some way. The process of preparing to run a callback and cleaning up after running a callback manipulate this stack. [WEBIDL]

When Web IDL is used to invoke author code, or when HostEnqueuePromiseJob invokes a promise job, they use the following algorithms to track relevant data for determining the incumbent settings object:

To prepare to run a callback with an environment settings object settings:

1. Push settings onto the backup incumbent settings object stack.
2. Let context be the topmost script-having execution context.
3. If context is not null, increment context's skip-when-determining-incumbent counter.

To clean up after running a callback with an environment settings object settings:

1. Let context be the topmost script-having execution context.
2. If context is not null, decrement context's skip-when-determining-incumbent counter.
3. Assert: the topmost entry of the backup incumbent settings object stack is settings.
4. Remove settings from the backup incumbent settings object stack.

Here, the topmost script-having execution context is the topmost entry of the JavaScript execution context stack that has a non-null ScriptOrModule component, or null if there is no such entry in the JavaScript execution context stack.

With all this in place, the incumbent settings object is determined as follows:

1. Let context be the topmost script-having execution context.
2. If context is null, or if context's skip-when-determining-incumbent counter is greater than zero, then:
   1. Assert the backup incumbent settings object stack is not empty.

Note: This will be the same as the topmost script-having execution context inside the corresponding invocation of prepare to run a callback.

This assert would fail if you try to obtain the incumbent settings object from inside an algorithm that was triggered neither by calling scripts nor by Web IDL invoking a callback. For example, it would trigger if you tried to obtain the incumbent settings object inside an algorithm that ran periodically as part of the event loop, with no involvement of author code. In such cases the incumbent concept cannot be used.
2. Return the topmost entry of the backup incumbent settings object stack.

3. Return context's Realm component's settings object.

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.

The following series of examples is intended to make it clear how all of the different mechanisms contribute to the definition of the incumbent concept:

Example
Consider the following starter example:

```html
<!DOCTYPE html>
<iframe></iframe>
<script>
  frames[0].postMessage("some data", ";");
</script>
```

There are two interesting environment settings objects here: that of window, and that of frames[0]. Our concern is: what is the incumbent settings object at the time that the algorithm for `postMessage()` executes?

It should be that of window, to capture the intuitive notion that the author script responsible for causing the algorithm to happen is executing in window, not frames[0]. This makes sense: the window post message steps use the incumbent settings object to determine the source property of the resulting MessageEvent, and in this case window is definitely the source of the message.

Let us now explain how the steps given above give us our intuitively-desired result of window's relevant settings object.

When the window post message steps look up the incumbent settings object, the topmost script-having execution context will be that corresponding to the script element: it was pushed onto the JavaScript execution context stack as part of ScriptEvaluation during the run a classic script algorithm. Since there are no Web IDL callback invocations involved, the context's skip-when-determining-incumbent counter is zero, so it is used to determine the incumbent settings object; the result is the environment settings object of window.

(Note how the environment settings object of frames[0] is the relevant settings object of this at the time the `postMessage()` method is called, and thus is involved in determining the target of the message. Whereas the incumbent is used to determine the source.)

Example
Consider the following more complicated example:

```html
<!DOCTYPE html>
<iframe></iframe>
<script>
  const bound = frames[0].postMessage.bind(frames[0], "some data", ";");
  window.setTimeout(bound);
</script>
```

This example is very similar to the previous one, but with an extra indirection through `Function.prototype.bind` as well as `setTimeout()`. But, the answer should be the same: invoking algorithms asynchronously should not change the incumbent concept.

This time, the result involves more complicated mechanisms:

When bound is converted to a Web IDL callback type, the incumbent settings object is that corresponding to window (in the same manner as in our starter example above). Web IDL stores this as the resulting callback value's callback context.

When the task posted by `setTimeout()` executes, the algorithm for that task uses Web IDL to invoke the stored callback.
value. Web IDL in turn calls the above prepare to run a callback\(^{907}\) algorithm. This pushes the stored callback context onto the backup incumbent settings object stack\(^{907}\). At this time (inside the timer task) there is no author code on the stack, so the topmost script-having execution context\(^{907}\) is null, and nothing gets its skip-when-determining-incumbent counter\(^{907}\) incremented.

Invoking the callback then calls bound, which in turn calls the \texttt{postMessage()\(^{1001}\)} method of frames[0]. When the \texttt{postMessage()\(^{1001}\)} algorithm looks up the incumbent settings object\(^{907}\), there is still no author code on the stack, since the bound function just directly calls the built-in method. So the topmost script-having execution context\(^{907}\) will be null: the \texttt{JavaScript execution context} stack only contains an execution context corresponding to \texttt{postMessage()\(^{1001}\)}, with no \texttt{ScriptEvaluation} context or similar below it.

This is where we fall back to the backup incumbent settings object stack\(^{907}\). As noted above, it will contain as its topmost entry the relevant settings object\(^{910}\) of \texttt{window}. So that is what is used as the incumbent settings object\(^{907}\) while executing the \texttt{postMessage()\(^{1001}\)} algorithm.

### Example

Consider this final, even more convoluted example:

```html
<!DOCTYPE html>
<!-- a.html -->
<script>
  const bound = frames[0].location.assign.bind(frames[0].location, "https://example.com/");
  document.querySelector("button").addEventListener("click", bound);
</script>

<!DOCTYPE html>
<!-- b.html -->
<script>
  const iframe = document.querySelector("iframe");
  iframe.onload = function onLoad() {
    iframe.contentWindow.document.querySelector("button").click();
  };
</script>
```

Again there are two interesting environment settings objects\(^{903}\) in play: that of a.html, and that of b.html. When the \texttt{location.assign()}\(^{1072}\) method triggers the Location-object navigate\(^{686}\) algorithm, what will be the incumbent settings object\(^{907}\)? As before, it should intuitively be that of a.html: the \texttt{click} listener was originally scheduled by a.html, so even if something involving b.html causes the listener to fire, the incumbent\(^{905}\) responsible is that of a.html.

The callback setup is similar to the previous example: when bound is converted to a Web IDL callback type, the incumbent settings object\(^{907}\) is that corresponding to a.html, which is stored as the callback's callback context.

When the \texttt{click()}\(^{772}\) method is called inside b.html, it dispatches a \texttt{click} event on the button that is inside a.html. This time, when the prepare to run a callback\(^{907}\) algorithm executes as part of event dispatch, there is author code on the stack; the topmost script-having execution context\(^{907}\) is that of the onLoad function, whose skip-when-determining-incumbent counter\(^{907}\) gets incremented. Additionally, a.html's environment settings object\(^{903}\) (stored as the \texttt{EventHandler}\(^{565}\)'s callback context) is pushed onto the backup incumbent settings object stack\(^{907}\).

Now, when the Location-object navigate\(^{686}\) algorithm looks up the incumbent settings object\(^{907}\), the topmost script-having execution context\(^{907}\) is still that of the \texttt{onLoad} function (due to the fact we are using a bound function as the callback). Its skip-when-determining-incumbent counter\(^{907}\) value is one, however, so we fall back to the backup incumbent settings object stack\(^{907}\).

This gives us the environment settings object\(^{903}\) of a.html, as expected.

Note that this means that even though it is the \texttt{iframe}\(^{282}\) inside a.html that navigates, it is a.html itself that is used as the source browsing context\(^{475}\), which determines among other things the request client. This is perhaps the only justifiable use of the incumbent concept on the web platform; in all other cases the consequences of using it are simply confusing and we hope to one day switch them to use current\(^{907}\) or relevant\(^{905}\) as appropriate.
### 8.1.3.3 Current

The JavaScript specification defines the current Realm Record, sometimes abbreviated to the "current Realm". 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.

### 8.1.3.4 Relevant

The relevant Realm for a platform object is the value of its [[Realm]] field.

Then, the relevant settings object for a platform object is the environment settings object of the relevant Realm for the object.

Similarly, the relevant global object for a platform object is the global object of the relevant Realm for the object.

### 8.1.3.4 Enabling and disabling scripting

**Scripting is enabled** for an environment settings object settings when all of the following conditions are true:

- The user agent supports scripting.
- The user has not disabled scripting for settings 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, down to the level of individual environment settings objects.)
- Either settings's global object is not a Window object, or settings's global object's associated Document's active sandboxing flag does not have its sandboxed scripts browsing context flag set.

**Scripting is disabled** for an environment settings object when scripting is not enabled for it, i.e., when any of the above conditions are false.

**Scripting is enabled** for a node if node's node document's browsing context is non-null, and scripting is enabled for node's relevant settings object.

**Scripting is disabled** for a node when scripting is not enabled, i.e., when its node document's browsing context is null or when scripting is disabled for its relevant settings object.

### 8.1.3.5 Secure contexts

An environment environment is a secure context if the following algorithm returns true:

1. If environment is an environment settings object, then:
   1. Let global be environment's global object.
   2. If global is a WorkerGlobalScope, then:
      1. If global's owner set's relevant settings object is a secure context, then return true.
      
      **Note**
      
      We only need to check the 0th item since they will necessarily all be consistent.

    2. Return false.

2. Return false.

3. If global is a WorkletGlobalScope, then return true.

**Note**

Worklets can only be created in secure contexts.
2. If the result of Is url potentially trustworthy? given environment's top-level creation URL is "Potentially Trustworthy", then return true.

3. Return false.

An environment is a non-secure context if it is not a secure context.

### 8.1.4 Script processing model

#### 8.1.4.1 Scripts

A script is one of two possible structs. All scripts have:

**A settings object**

An environment settings object, containing various settings that are shared with other scripts in the same context.

**A record**

Either a Script Record, for classic scripts; a Source Text Module Record, for module scripts; or null. In the former two cases, it represents a parsed script; null represents a failure parsing.

**A parse error**

A JavaScript value, which has meaning only if the record is null, indicating that the corresponding script source text could not be parsed.

**Note**

This value is used for internal tracking of immediate parse errors when creating scripts, and is not to be used directly. Instead, consult the error to rethrow when determining "what went wrong" for this script.

**An error to rethrow**

A JavaScript value representing an error that will prevent evaluation from succeeding. It will be re-thrown by any attempts to run the script.

**Note**

This could be the script's parse error, but in the case of a module script it could instead be the parse error from one of its dependencies, or an error from resolve a module specifier.

**Note**

Since this exception value is provided by the JavaScript specification, we know that it is never null, so we use null to signal that no error has occurred.

**Fetch options**

A script fetch options, containing various options related to fetching this script or module scripts that it imports.

**A base URL**

A base URL used for resolving module specifiers. This will either be the URL from which the script was obtained, for external scripts, or the document base URL of the containing document, for inline scripts.

**A classic script** is a type of script that has the following additional item:

**A muted errors boolean**

A boolean which, if true, means that error information will not be provided for errors in this script. This is used to mute errors for cross-origin scripts, since that can leak private information.

**A module script** is another type of script. It has no additional items.

The active script is determined by the following algorithm:

1. Let record be GetActiveScriptOrModule().
2. If record is null, return null.
3. Return record. [[HostDefined]].

### 8.1.4.2 Fetching scripts

This section introduces a number of algorithms for fetching scripts, taking various necessary inputs and resulting in classic or module scripts.

**Script fetch options** is a struct with the following items:

- **cryptographic nonce**
  - The cryptographic nonce metadata used for the initial fetch and for fetching any imported modules

- **integrity metadata**
  - The integrity metadata used for the initial fetch

- **parser metadata**
  - The parser metadata used for the initial fetch and for fetching any imported modules

- **credentials mode**
  - The credentials mode used for the initial fetch (for module scripts and classic scripts) and for fetching any imported modules (for both module scripts and classic scripts)

- **referrer policy**
  - The referrer policy used for the initial fetch and for fetching any imported modules

**Note**

Recall that via the import() feature, classic scripts can import module scripts.

The default classic script fetch options are a script fetch options whose cryptographic nonce is the empty string, integrity metadata is the empty string, parser metadata is "not-parser-inserted", credentials mode is "same-origin", and referrer policy is the empty string.

Given a request request and a script fetch options options, we define:

- **set up the classic script request**
  - Set request's cryptographic nonce metadata to options's cryptographic nonce, its integrity metadata to options's integrity metadata, its parser metadata to options's parser metadata, and its referrer policy to options's referrer policy.

- **set up the module script request**
  - Set request's cryptographic nonce metadata to options's cryptographic nonce, its integrity metadata to options's integrity metadata, its parser metadata to options's parser metadata, its credentials mode to options's credentials mode, and its referrer policy to options's referrer policy.

For any given script fetch options options, the descendant script fetch options are a new script fetch options whose items all have the same values, except for the integrity metadata, which is instead the empty string.

The algorithms below can be customized by optionally supplying a custom perform the fetch hook, which takes a request and an is top-level flag. The algorithm must complete with a response (which may be a network error), either synchronously (when using fetch a classic worker-imported script or asynchronously (otherwise). The is top-level flag will be set for all classic script fetches, and for the initial fetch when fetching an external module script graph, fetching a module worker script graph, or fetching an import() module script graph, but not for the fetches resulting from import statements encountered throughout the graph.
Now for the algorithms themselves.

To fetch a classic script given a url, a settings object, some options, a CORS setting, 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, "script", and CORS setting.
2. Set request's client to settings object.
3. Set up the classic script request given request and options.
4. If the caller specified custom steps to perform the fetch, perform them on request, with the is top-level flag set. Return from this algorithm, and when the custom perform the fetch steps complete with response response, run the remaining steps.
   Otherwise, fetch request. Return from this algorithm, and run the remaining steps as part of the fetch's process response for the response response.
5. Set response to response's unsafe response.
6. If response's type is "error", or response's status is not an ok status, then asynchronously complete this algorithm with null, and return.
7. 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).
8. Let source text be the result of decoding response's body to Unicode, using character encoding as the fallback encoding.
9. Let muted errors be true if response was CORS-cross-origin, and false otherwise.
10. Let script be the result of creating a classic script given source text, settings object, response's url, options, and muted errors.
11. Asynchronously complete this algorithm with script.

To fetch a classic worker script given a url, a fetch client settings object, a destination, and a script settings object, 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 fetch client settings object, destination is destination, 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 perform the fetch, perform them on request, with the is top-level flag set. Return from this algorithm, and when the custom perform the fetch steps complete with response, run the remaining steps.

Otherwise, fetch request. Return from this algorithm, and run the remaining steps as part of the fetch's process response for the response.

3. Set response to response's unsafe response.

4. If either of the following conditions are met:
   - response's type is "error"; or
   - response's status is not an ok status,

   then asynchronously complete this algorithm with null, and return.

5. If both of the following conditions are met:
   - response's url's scheme is an HTTP(S) scheme; and
   - the result of extracting a MIME type from response's header list is not a JavaScript MIME type,

   then asynchronously complete this algorithm with null, and return.

Note
Other fetch schemes are exempted from MIME type checking for historical web-compatibility reasons. We might be able to tighten this in the future; see issue #3255.

6. Let source text be the result of UTF-8 decoding response's body.

7. Let script be the result of creating a classic script using source text, script settings object, response's url, and the default classic script fetch options.

8. Asynchronously complete this algorithm with script.

To fetch a classic worker-imported script given a url and a settings object, run these steps. The algorithm will synchronously complete with a classic script on success, or throw an exception on failure.

1. Let request be a new request whose url is url, client is settings object, destination is "script", parser metadata is "not parser-inserted", synchronous flag is set, and whose use-URL-credentials flag is set.

2. If the caller specified custom steps to perform the fetch, perform them on request, with the is top-level flag set. Let response be the result.

Otherwise, fetch request, and let response be the result.

Note
Unlike other algorithms in this section, the fetching process is synchronous here. Thus any perform the fetch steps will also finish their work synchronously.

3. Set response to response's unsafe response.

4. If any of the following conditions are met:
   - response's type is "error"; or
   - response's status is not an ok status; or
   - the result of extracting a MIME type from response's header list is not a JavaScript MIME type,

   then throw a "NetworkError" DOMException.

5. Let source text be the result of UTF-8 decoding response's body.

6. Let muted errors be true if response was CORS-cross-origin, and false otherwise.
7. Let script be the result of creating a classic script given source text, settings object, response's url, the default classic script fetch options, and muted errors.

8. Return script.

To fetch an external module script graph given a url, a settings object, and some options, run these steps. The algorithm will asynchronously complete with either null (on failure) or a module script (on success).

1. Fetch a single module script given url, settings object, "script", options, settings object, "client", and with the top-level module fetch flag set. If the caller of this algorithm specified custom perform the fetch steps, pass those along as well. Wait until the algorithm asynchronously completes with result.

2. If result is null, asynchronously complete this algorithm with null, and return.

3. Let visited set be « url ».

4. Fetch the descendants of and link result given settings object, destination, and visited set. When this asynchronously completes with final result, asynchronously complete this algorithm with final result.

To fetch an import() module script graph given a specifier, a base URL, a settings object, and some options, run these steps. The algorithm will asynchronously complete with either null (on failure) or a module script (on success).

1. Let url be the result of resolving a module specifier given base URL and specifier.

2. If url is failure, then asynchronously complete this algorithm with null, and return.

3. Fetch a single module script given url, settings object, "script", options, settings object, "client", and with the top-level module fetch flag set. If the caller of this algorithm specified custom perform the fetch steps, pass those along as well. Wait until the algorithm asynchronously completes with result.

4. If result is null, asynchronously complete this algorithm with null, and return.

5. Let visited set be « url ».

6. Fetch the descendants of and link result given settings object, destination, and visited set. When this asynchronously completes with final result, asynchronously complete this algorithm with final result.

To fetch a modulepreload module script graph given a url, a destination, a settings object, and some options, run these steps. The algorithm will asynchronously complete with either null (on failure) or a module script (on success), although it will perform optional steps even after completing.

1. Fetch a single module script given url, settings object, destination, options, settings object, "client", and with the top-level module fetch flag set. Wait until algorithm asynchronously completes with result.

2. Asynchronously complete this algorithm with result, but do not return.

3. If result is not null, optionally perform the following steps:
   1. Let visited set be « url ».
   2. Fetch the descendants of and link result given settings object, destination, and visited set.

Note Generally, performing these steps will be beneficial for performance, as it allows pre-loading the modules that will invariably be requested later, via algorithms such as fetch an external module script graph that fetch the entire graph. However, user agents might wish to skip them in bandwidth-constrained situations, or situations where the relevant fetches are already in flight.

To fetch an inline module script graph given a source text, base URL, settings object, and options, run these steps. The algorithm will asynchronously complete with either null (on failure) or a module script (on success).

1. Let script be the result of creating a module script using source text, settings object, base URL, and options.

2. If script is null, asynchronously complete this algorithm with null, and return.

3. Let visited set be an empty set.

4. Fetch the descendants of and link script, given settings object, the destination "script", and visited set. When this
To **fetch a module worker script graph** given a url, a fetch client settings object, a destination, a credentials mode, and a module map settings object, fetch a worklet/module worker script graph given url, fetch client settings object, destination, credentials mode, and module map settings object, asynchronously completing with the asynchronous completion result of that algorithm.

To **fetch a worklet script graph** given a url, a fetch client settings object, a destination, a credentials mode, a module map settings object, and a module responses map, fetch a worklet/module worker script graph given url, fetch client settings object, destination, credentials mode, and module map settings object, asynchronously completing with the asynchronous completion result of that algorithm.

The following algorithms are meant for internal use by this specification only as part of fetching an external module script graph or other similar concepts above, and should not be used directly by other specifications.

This diagram illustrates how these algorithms relate to the ones above, as well as to each other:

---

To **fetch a worklet/module worker script graph** given a url, a fetch client settings object, a destination, a credentials mode, and a module map settings object, run these steps. The algorithm will asynchronously complete with either null (on failure) or a module script (on success).

1. Let options be a script fetch options whose cryptographic nonce is the empty string, integrity metadata is the empty string, parser metadata is "not-parser-inserted", credentials mode is credentials mode, and referrer policy is the empty string.
2. Fetch a single module script given url, fetch client settings object, destination, options, module map settings object, "client", and with the top-level module fetch flag set. If the caller of this algorithm specified custom perform the fetch steps, pass those along as well. Wait until the algorithm asynchronously completes with result.
3. If result is null, asynchronously complete this algorithm with null, and return.
4. Let visited set be « url ».
5. Fetch the descendants of and link a module script result given fetch client settings object, destination, and visited set. When this asynchronously completes with final result, asynchronously complete this algorithm with final result.

To **fetch the descendants of and link a module script** module script, given a fetch client settings object, a destination, and a visited set, run these steps. The algorithm will asynchronously complete with either null (on failure) or with module script (on success).
1. Fetch the descendants of module script, given fetch client settings object, destination, and visited set.

2. Return from this algorithm, and run the following steps when fetching the descendants of a module script asynchronously completes with result.

3. If result is null, then asynchronously complete this algorithm with result.

   **Note**
   In this case, there was an error fetching one or more of the descendants. We will not attempt to link.

4. Let parse error be the result of finding the first parse error, given result.

5. If parse error is null, then:
   1. Let record be result's record.
   2. Perform record.Link()

   **Note**
   This step will recursively call Link on all of the module's unlinked dependencies.

   If this throws an exception, set result's error to rethrow to that exception.

6. Otherwise, set result's error to rethrow to parse error.

7. Asynchronously complete this algorithm with result.

To fetch the descendants of a module script module script, given a fetch client settings object, a destination, and a visited set, run these steps. The algorithm will asynchronously complete with either null (on failure) or with module script (on success).

1. If module script's record is null, then asynchronously complete this algorithm with module script and return.

2. Let record be module script's record.

3. If record is not a Cyclic Module Record, or if record.[[RequestedModules]] is empty, asynchronously complete this algorithm with module script.

4. Let urls be a new empty list.

5. For each string requested of record.[[RequestedModules]],
   1. Let url be the result of resolving a module specifier given module script's base URL and requested.
   2. Assert: url is never failure, because resolving a module specifier must have been previously successful with these same two arguments.
   3. If visited set does not contain url, then:
      1. Append url to urls.
      2. Append url to visited set.

6. Let options be the descendant script fetch options for module script's fetch options.

7. Assert: options is not null, as module script is a module script.

8. For each url in urls, perform the internal module script graph fetching procedure given url, fetch client settings object, destination, options, module script's settings object, visited set, and module script's base URL. If the caller of this algorithm specified custom perform the fetch steps, pass those along while performing the internal module script graph fetching procedure.

These invocations of the internal module script graph fetching procedure should be performed in parallel to each other.

If any of the invocations of the internal module script graph fetching procedure asynchronously complete with null, asynchronously complete this algorithm with null, and return.

Otherwise, wait until all of the internal module script graph fetching procedure invocations have asynchronously completed. Asynchronously complete this algorithm with module script.
To perform the **internal module script graph fetching procedure** given a `url`, a `fetch client settings object`, a `destination`, some `options`, a `module map settings object`, a `visited set`, and a `referrer`, perform these steps. The algorithm will asynchronously complete with either null (on failure) or a `module script` (on success).

1. **Assert:** `visited set contains url`.
2. **Fetch a single module script** given `url`, `fetch client settings object`, `destination`, `options`, `module map settings object`, `referrer`, and with the top-level module fetch flag unset. If the caller of this algorithm specified custom perform the fetch steps, pass those along while fetching a single module script.
3. Return from this algorithm, and run the following steps when fetching a single module script asynchronously completes with result:
   4. If result is null, asynchronously complete this algorithm with null, and return.
   5. Fetch the descendants of result given `fetch client settings object`, `destination`, and `visited set`.
   6. When the appropriate algorithm asynchronously completes with final result, asynchronously complete this algorithm with final result.

To **fetch a single module script**, given a `url`, a `fetch client settings object`, a `destination`, some `options`, a `module map settings object`, a `referrer`, and a top-level module fetch flag, run these steps. The algorithm will asynchronously complete with either null (on failure) or a `module script` (on success).

1. Let `moduleMap` be `module map`'s `module map`.
2. If `moduleMap[url]` is "fetching", wait in parallel until that entry's value changes, then queue a task on the networking task source to proceed with running the following steps.
3. If `moduleMap[url]` exists, asynchronously complete this algorithm with `moduleMap[url]`, and return.
4. Set `moduleMap[url]` to "fetching".
5. Let `request` be a new request whose `url` is `url`, `destination` is `destination`, `mode` is "cors", `referrer` is `referrer`, and `client` is `fetch client settings object`.
6. If `destination` is "worker" or "sharedworker" and the top-level module fetch flag is set, then set request's `mode` to "same-origin".
7. Set up the module script request given `request` and `options`.
8. If the caller specified custom steps to perform the fetch, perform them on request, setting the is top-level flag if the top-level module fetch flag is set. Return from this algorithm, and when the custom perform the fetch steps complete with response, run the remaining steps.
    
    Otherwise, fetch request. Return from this algorithm, and run the remaining steps as part of the fetch's process response for the response.

    **Note:** response is always CORS-same-origin.

9. If any of the following conditions are met:
   4. response's `type` is "error"; or
   5. response's `status` is not an ok status; or
   6. the result of extracting a MIME type from response's header list is not a JavaScript MIME type.

    then set `moduleMap[url]` to null, asynchronously complete this algorithm with null, and return.
10. Let `source text` be the result of UTF-8 decoding response's body.
11. Let `module script` be the result of creating a module script given `source text`, `module map settings object`, response's `url`, and `options`.
12. Set `moduleMap[url]` to `module script`, and asynchronously complete this algorithm with `module script`.

918
To find the first parse error given a root moduleScript and an optional discoveredSet:

1. Let moduleMap be moduleScript’s settings object’s module map.
2. If discoveredSet was not given, let it be an empty set.
3. Append moduleScript to discoveredSet.
4. If moduleScript’s record is null, then return moduleScript’s parse error.
5. Let childSpecifiers be the value of moduleScript’s record’s [[RequestedModules]] internal slot.
6. Let childURLs be the list obtained by calling resolve a module specifier once for each item of childSpecifiers, given moduleScript’s base URL and that item. (None of these will ever fail, as otherwise moduleScript would have been marked as itself having a parse error.)
7. Let childModules be the list obtained by getting each value in moduleMap whose key is given by an item of childURLs.
8. For each childModule of childModules:
   1. Assert: childModule is a module script (i.e., it is not "fetching" or null); by now all module scripts in the graph rooted at moduleScript will have successfully been fetched.
   2. If discoveredSet already contains childModule, continue.
   3. Let childParseError be the result of finding the first parse error given childModule and discoveredSet.
   4. If childParseError is not null, return childParseError.

8.1.4.3 Creating scripts

To create a classic script, given a string source, an environment settings object’s settings, a URL base URL, some script fetch options options and an optional muted errors boolean:

1. If muted errors was not provided, let it be false.
2. If muted errors is true, then set base URL to about:blank.
3. If scripting is disabled for settings, then set source to the empty string.
4. Let script be a new classic script that this algorithm will subsequently initialize.
5. Set script’s settings object to settings.
6. Set script’s base URL to base URL.
7. Set script’s fetch options to options.
8. Set script’s muted errors to muted errors.
9. Set script’s parse error and error to rthrow to null.
10. Let result be ParseScript(source, settings’s Realm, script).

Note

When muted errors is true, base URL is the script’s CORS-cross-origin response’s url, which shouldn’t be exposed to JavaScript. Therefore, base URL is sanitized here.
To create a module script, given a string source, an environment settings object, a URL, and some script fetch options:

1. If scripting is disabled for settings, then set source to the empty string.
2. Let script be a new module script that this algorithm will subsequently initialize.
3. Set script’s settings object to settings.
4. Set script’s base URL to baseURL.
5. Set script’s fetch options to options.
6. Set script’s parse error and error to rethrow to null.
7. Let result be ParseModule(source, settings's Realm, script).
8. If result is a list of errors, then:
   1. Set script’s parse error and error to rethrow to result[0].
   2. Return script.
9. For each string requested of result.[RequestedModules]:
   1. Let url be the result of resolving a module specifier given script’s base URL and requested.
   2. If url is failure, then:
      1. Let error be a new TypeError exception.
      2. Set script’s parse error and error to rethrow to error.
      3. Return script.
10. Set script’s record to result.
11. Return script.

To run a classic script given a classic script script and an optional boolean rethrow errors (default false):

1. Let settings be the settings object of script.
2. Check if we can run script with settings. If this returns “do not run” then return NormalCompletion(empty).
3. Prepare to run script given settings.
4. Let evaluationStatus be null.

8.1.4.4 Calling scripts

To create a module script, given a string source, an environment settings object, a URL, and some script fetch options:
5. If script's error to rethrow is not null, then set evaluationStatus to Completion { [[Type]]: throw, [[Value]]: script's error to rethrow, [[Target]]: empty }.

6. Otherwise, set evaluationStatus to ScriptEvaluation(script's record).

   If ScriptEvaluation does not complete because the user agent has aborted the running script, leave evaluationStatus as null.

7. If evaluationStatus is an abrupt completion, then:

   1. If rethrow errors is true and script's muted errors is false, then:
      1. Clean up after running script with settings.
      2. Rethrow evaluationStatus.[Value].
   2. If rethrow errors is true and script's muted errors is true, then:
      1. Clean up after running script with settings.
      2. Throw a "NetworkError" DOMException.
   3. Otherwise, rethrow errors is false. Perform the following steps:
      1. Report the exception given by evaluationStatus.[Value] for script.
      2. Clean up after running script with settings.
      3. Return evaluationStatus.

8. Clean up after running script with settings.

9. If evaluationStatus is a normal completion, then return evaluationStatus.

10. If we've reached this point, evaluationStatus was left as null because the script was aborted prematurely during evaluation. Return Completion { [[Type]]: throw, [[Value]]: a new "QuotaExceededError" DOMException, [[Target]]: empty }.

To run a module script given a module script script and an optional boolean preventErrorReporting (default false):

1. Let settings be the settings object of script.
2. Check if we can run script with settings. If this returns "do not run", then return a promise resolved with undefined.
3. Prepare to run script given settings.
4. Let evaluationPromise be null.
5. If script's error to rethrow is not null, then set evaluationPromise to a promise rejected with script's error to rethrow.
6. Otherwise:

   1. Let record be script's record.
   2. Set evaluationPromise to record.Evaluate().

   Note: This step will recursively evaluate all of the module's dependencies.

   If Evaluate fails to complete as a result of the user agent aborting the running script, then set evaluationPromise to a promise rejected with a new "QuotaExceededError" DOMException.

7. If preventErrorReporting is false, then upon rejection of evaluationPromise with reason, report the exception given by reason for script.
8. Clean up after running script with settings.
9. Return evaluationPromise.

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".

2. If scripting is disabled for settings, then return "do not run".

3. Return "run".

The steps to prepare to run script with an environment settings object settings are as follows:

1. Push settings's realm execution context onto the JavaScript execution context stack; it is now the running JavaScript execution context.

2. Add settings to the currently running task's script evaluation environment settings object set.

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. Remove settings's realm execution context from the JavaScript execution context stack.

3. 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 ScriptOrModule component of the running JavaScript execution context.

### 8.1.4.5 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 or Source Text Module Record Evaluate invocations to cease immediately, emptying the JavaScript execution context stack without triggering any of the normal mechanisms like finally blocks. [JAVASCRIPT]

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" DOMException, abort the script without an exception, prompt the user, or throttle script execution.

**Example**

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.

```html
<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**

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.
When the user agent is required to report an error for a particular `script` 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 return; the error is not handled.
2. Let `target` be in error reporting mode.
3. Let `message` be an implementation-defined string describing the error in a helpful manner.
4. Let `errorValue` be the value that represents the error: in the case of an uncaught exception, that would be the value that was thrown; in the case of a JavaScript error that would be an `Error` object. If there is no corresponding value, then the null value must be used instead.
5. Let `urlString` be the result of applying the URL serializer to the `URL record` that corresponds to the resource from which `script` was obtained.
6. If `script`'s muted errors is true, then set `message` to "Script error.", `urlString` to the empty string, `line` and `col` to 0, and `errorValue` to null.
7. Let `notHandled` be true.
8. If `target` implements `EventTarget`, then set `notHandled` to the result of firing an event named `error` at `target`, using `ErrorEvent`, with the `cancelable` attribute initialized to true, the `message` attribute initialized to `message`, the `filename` attribute initialized to `urlString`, the `lineno` attribute initialized to `line`, the `colno` attribute initialized to `col`, and the `error` attribute initialized to `errorValue`.
9. Let `target` no longer be in error reporting mode.
10. If `notHandled` is false, then the error is handled. Otherwise, the error is not handled.

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.

The existence of both report an error and report an exception is confusing, and both algorithms have known problems. You can track future cleanup in this area in issue #958.

The `ErrorEvent` interface is defined as follows:

```idl
[Exposed=(Window,Worker)]
interface ErrorEvent : Event {
    constructor(DOMString type, optional ErrorEventInit eventInitDict = {});
}
```
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 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).

8.1.4.7 Unhandled promise rejections

In addition to synchronous runtime script errors, scripts may experience asynchronous promise rejections, tracked via the unhandledrejection and rejectionhandled events. Tracking these rejections is done via the HostPromiseRejectionTracker abstract operation, but reporting them is defined here.

To notify about rejected promises on a given environment settings object settings object:

1. Let list be a copy of settings object's about-to-be-notified rejected promises list.
2. If list is empty, return.
4. Let global be settings object's global object.
5. Queue a global task on the DOM manipulation task source given global to run the following substep:
   1. For each promise p in list:
      1. If p's [[PromisesHandled]] internal slot is true, continue to the next iteration of the loop.
      2. Let notHandled be the result of firing an event named unhandledrejection at global, using PromiseRejectionEvent, with the cancelable attribute initialized to true, the promise attribute initialized to p, and the reason attribute initialized to the value of p's [[PromiseResult]] internal slot.
      3. If notHandled is false, then the promise rejection is handled. Otherwise, the promise rejection is not handled.
      4. If p's [[PromisesHandled]] 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.

The PromiseRejectionEvent interface is defined as follows:
IDL

```idl
[Exposed=(Window,Worker)]
interface PromiseRejectionEvent : Event {
    constructor(DOMString type, PromiseRejectionEventInit eventInitDict);
    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.

### 8.1.5 JavaScript specification host hooks

The JavaScript specification contains a number of implementation-defined abstract operations, that vary depending on the host environment. This section defines them for user agent hosts.

#### 8.1.5.1 HostEnsureCanCompileStrings(callerRealm, calleeRealm)

JavaScript contains an implementation-defined `HostEnsureCanCompileStrings(callerRealm, calleeRealm)` abstract operation. User agents must use the following implementation: [JAVASCRIPT][1273]

1. Perform ? EnsureCSPDoesNotBlockStringCompilation(callerRealm, calleeRealm). [CSP][1279]

#### 8.1.5.2 HostPromiseRejectionTracker(promise, operation)

JavaScript contains an implementation-defined `HostPromiseRejectionTracker(promise, operation)` abstract operation. User agents must use the following implementation: [JAVASCRIPT][1273]

1. Let `script` be the running script.
2. If `script`'s `muted errors` is true, 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`, then remove `promise` from that list and return.
2. If `settings object`'s `outstanding rejected promises weak set` does not contain `promise`, then return.
3. Remove `promise` from `settings object`'s `outstanding rejected promises weak set`.
4. Let `global` be `settings object`'s `global object`.
5. Queue a global task on the DOM manipulation task source given `global` to fire an event named `rejectionhandled` at `global`, using `PromiseRejectionEvent`, with the `promise` attribute initialized to `promise`, and the `reason` attribute initialized to the value of `promise`'s `[[PromiseResult]]` internal slot.
8.1.5.3 Job-related host hooks §921

The JavaScript specification defines Jobs to be scheduled and run later by the host, as well as JobCallback Records which encapsulate JavaScript functions that are called as part of jobs. The JavaScript specification contains a number of implementation-defined abstract operations that lets the host define how jobs are scheduled and how JobCallbacks are handled. This section defines them for user agent hosts.

8.1.5.3.1 HostCallJobCallback(callback, V, argumentsList) §921

JavaScript contains an implementation-defined HostCallJobCallback(callback, V, argumentsList) abstract operation to let hosts restore state when invoking JavaScript callbacks from inside tasks. HTML restores the incumbent settings object§907 and the active script§911. User agents must use the following implementation: [JAVASCRIPT]§1273

1. Let incumbent settings be callback.[[HostDefined]].[[IncumbentSettings]].
2. Let script execution context be callback.[[HostDefined]].[[ActiveScriptContext]].
3. Prepare to run a callback§907 with incumbent settings.
   
   Note
   This affects the incumbent§995 concept while the callback runs.

4. If script execution context is not null, then push script execution context onto the JavaScript execution context stack.
   
   Note
   This affects the active script§921 while the callback runs.

5. Let result be Call(callback.[[Callback]], V, argumentsList).
6. If script execution context is not null, then pop script execution context from the JavaScript execution context stack.
7. Clean up after running a callback§907 with incumbent settings.
8. Return result.

8.1.5.3.2 HostEnqueuePromiseJob(job, realm) §921

JavaScript contains an implementation-defined HostEnqueuePromiseJob(job, realm) abstract operation to schedule Promise-related operations. HTML schedules these operations in the microtask queue. User agents must use the following implementation: [JAVASCRIPT]§1273

1. If realm is not null, then let job settings be the settings object§904 for realm. Otherwise, let job settings be null.
   
   Note
   If realm is not null, it is the Realm of the author code that will run. When job is returned by NewPromiseReactionJob, it is the realm of the promise's handler function. When job is returned by NewPromiseResolveThenableJob, it is the realm of the then function.
   
   If realm is null, either no author code will run or author code is guaranteed to throw. For the former, the author may not have passed in code to run, such as in promise.then(null, null). For the latter, it is because a revoked Proxy was passed. In both cases, all the steps below that would otherwise use job settings get skipped.

2. Queue a microtask§993 on the surrounding agent’s event loop§993 to perform the following steps:
   
   1. If job settings is not null, then check if we can run script§921 with job settings. If this returns “do not run” then return.
   2. If job settings is not null, then prepare to run script§922 with job settings.
   
   Note
   This affects the entry§995 concept while the job runs.
3. Let result be `job()`.

4. If `job settings` is not null, then clean up after running script with `job settings`.

5. If `result` is an abrupt completion, then report the exception given by `result.[Value]`.

---

**Note**

job is an abstract closure returned by `NewPromiseReactionJob` or `NewPromiseResolveThenableJob`. The promise's handler function when job is returned by `NewPromiseReactionJob`, and the then function when job is returned by `NewPromiseResolveThenableJob`, are wrapped in JobCallback Records. HTML saves the incumbent settings object and a JavaScript execution context for the active script in HostMakeJobCallback and restores them in HostCallJobCallback.

---

### 8.1.5.3.3 HostMakeJobCallback(callable)

JavaScript contains an implementation-defined `HostMakeJobCallback` (callable) abstract operation to let hosts attach state to JavaScript callbacks that are called from inside task's. HTML tracks the incumbent settings object in promises by saving the incumbent settings object and a JavaScript execution context for the active script. User agents must use the following implementation:

```javascript
1. Let incumbent settings be the incumbent settings object.
2. Let active script be the active script.
3. Let script execution context be null.
4. If active script is not null, set script execution context to a new JavaScript execution context, with its Function field set to null, its Realm field set to active script's settings object's Realm, and its ScriptOrModule set to active script's record.
5. Return the JobCallback Record {
    [[Callback]]: callable, 
    [[HostDefined]]: {
        [[IncumbentSettings]]: incumbent settings, 
        [[ActiveScriptContext]]: script execution context
    }
}
```

---

**Note**

As seen below, this is used in order to propagate the current active script forward to the time when the job callback is invoked.

---

**Example**

A case where active script is non-null, and saving it in this way is useful, is the following:

```javascript
Promise.resolve('import(`./example.mjs`)').then(eval);
```

Without this step (and the steps that use it in `HostCallJobCallback`), there would be no active script when the import() expression is evaluated, since eval() is a built-in function that does not originate from any particular script.

With this step in place, the active script is propagated from the above code into the job, allowing import() to use the original script's base URL appropriately.

**Example**

active script can be null if the user clicks on the following button:

```html
<button onclick="Promise.resolve('import(`./example.mjs`)').then(eval)">Click me</button>
```

In this case, the JavaScript function for the event handler will be created by the get the current value of the event handler algorithm, which creates a function with null [[ScriptOrModule]] value. Thus, when the promise machinery calls `HostMakeJobCallback`, there will be no active script to pass along.

As a consequence, this means that when the import() expression is evaluated, there will still be no active script. Fortunately that is handled by our implementations of `HostResolveImportedModule` and `HostImportModuleDynamically`, by falling back to using the current settings object's API base URL.

---

5. Return the JobCallback Record { [[Callback]]: callable, [[HostDefined]]: { [[IncumbentSettings]]: incumbent settings, [[ActiveScriptContext]]: script execution context } }.
The JavaScript specification defines a syntax for modules, as well as some host-agnostic parts of their processing model. This specification defines the rest of their processing model: how the module system is bootstrapped, via the `script` element with `type` attribute set to "module", and how modules are fetched, resolved, and executed.

### Note

Although the JavaScript specification speaks in terms of "scripts" versus "modules", in general this specification speaks in terms of classic scripts versus module scripts, since both of them use the `script` element.

#### For web developers (non-normative)

**modulePromise = `import(specifier)`**

Returns a promise for the module namespace object for the module script identified by specifier. This allows dynamic importing of module scripts at runtime, instead of statically using the `import` statement form. The specifier will be resolved relative to the `active script`'s base URL.

The returned promise will be rejected if an invalid specifier is given, or if a failure is encountered while fetching or evaluating the resulting module graph.

This syntax can be used inside both classic and module scripts, it thus provides a bridge into the module-script world, from the classic-script world.

**url = `import.meta.url`**

Returns the `active module script`'s base URL.

This syntax can only be used inside module scripts.

---

A module map is a map of URL records to values that are either a module script, null (used to represent failed fetches), or a placeholder value "fetching", Module maps are used to ensure that imported JavaScript modules are only fetched, parsed, and evaluated once per Document or worker.

**Example**

Since module maps are keyed by URL, the following code will create three separate entries in the module map, since it results in three different URLs:

```javascript
import "https://example.com/module.mjs";
import "https://example.com/module.mjs#map-buster";
import "https://example.com/module.mjs?debug=true";
```

That is, URL queries and fragments can be varied to create distinct entries in the module map; they are not ignored. Thus, three separate fetches and three separate module evaluations will be performed.

In contrast, the following code would only create a single entry in the module map, since after applying the URL parser to these inputs, the resulting URL records are equal:

```javascript
import "https://example.com/module2.mjs";
import "https://example.com/module2.mjs";
import "https://example.com\module2.mjs";
import "https://example.com/foo/../module2.mjs";
```

So in this second example, only one fetch and one module evaluation will occur.

Note that this behavior is the same as how shared workers are keyed by their parsed constructor url.

---

To resolve a module specifier given a URL base URL and a string specifier, perform the following steps. It will return either a URL record or failure.

1. Apply the URL parser to specifier. If the result is not failure, return the result.

2. If specifier does not start with the character U+002F SOLIDUS (/), the two-character sequence U+002E FULL STOP, U+002F SOLIDUS (./), or the three-character sequence U+002E FULL STOP, U+002E FULL STOP, U+002F SOLIDUS (../), return failure.
3. Return the result of applying the URL parser to `specifier` with base URL.

Example

The following are valid module specifiers according to the above algorithm:

- `https://example.com/apples.mjs`
- `http://example.com\pears.js` (becomes `http://example.com/pears.js` as step 1 parses with no base URL)
- `//example.com/bananas`
- `./strawberries.mjs.cgi`
- `../lychees`
- `//example.js`
- `data:text/javascript,export default 'grapes';`
- `blob:https://whatwg.org/d0360e2f-caee-469f-9a2f-87d5b0456f6f`

The following are valid module specifiers according to the above algorithm, but will invariably cause failures when they are fetched:

- `javascript:export default 'artichokes';`
- `data:text/plain,export default 'kale';`
- `about:legumes`
- `wss://example.com/celery`

The following are not valid module specifiers according to the above algorithm:

- `https://eggplant:b/c`
- `pumpkins.js`
- `..tomato`
- `..zucchini.mjs`
- `..yam.es`

### 8.1.5.4.1 HostGetImportMetaProperties(moduleRecord)

JavaScript contains an implementation-defined HostGetImportMetaProperties abstract operation. User agents must use the following implementation: [JAVASCRIPT]

1. Let `module script` be `moduleRecord`'s `[[HostDefined]]`.
2. Let `urlString` be `module script`'s base URL, serialized.
3. Return « Record { [[Key]]: "url", [[Value]]: `urlString` } ».

### 8.1.5.4.2 HostImportModuleDynamically(referencingScriptOrModule, specifier, promiseCapability)

JavaScript contains an implementation-defined HostImportModuleDynamically abstract operation. User agents must use the following implementation: [JAVASCRIPT]

1. Let `settings object` be the current settings object.
2. If `settings object`'s global object implements WorkletGlobalScope, then:
   1. Let `completion` be Completion { [[Type]]: throw, [[Value]]: a new TypeError, [[Target]]: empty }.
   2. Perform FinishDynamicImport(referencingScriptOrModule, specifier, promiseCapability, completion).
   3. Return.
3. Let base URL be `settings object`'s API base URL.
4. Let `fetch options` be the default classic script fetch options.
5. If referencingScriptOrModule is not null, then:
1. Let referencing script be referencingScriptOrModule.[[HostDefined]].
2. Set settings object to referencing script's settings object\(^{911}\).
3. Set base URL to referencing script's base URL\(^{911}\).
4. Set fetch options to the descendant script fetch options\(^{912}\) for referencing script's fetch options\(^{911}\).

**Note**

As explained above for HostResolveImportedModule\(^{920}\), in the common case, referencingScriptOrModule is non-null.

6. Fetch an import() module script graph\(^{915}\) given specifier, base URL, settings object, and fetch options. Wait until the algorithm asynchronously completes with result.

7. Let promise be null.

8. If result is null, then set promise to a promise rejected with a new TypeError.

9. Otherwise, set promise to the result of running a module script\(^{921}\) given result and true.


11. Return undefined.

8.1.5.4.3 HostResolveImportedModule(referencingScriptOrModule, specifier)\(^{930}\)

JavaScript contains an implementation-defined HostResolveImportedModule abstract operation. User agents must use the following implementation: [JAVASCRIPT]\(^{1273}\)

1. Let settings object be the current settings object\(^{910}\).
2. Let base URL be settings object's API base URL\(^{903}\).
3. If referencingScriptOrModule is not null, then:
   1. Let referencing script be referencingScriptOrModule.[[HostDefined]].
   2. Set settings object to referencing script's settings object\(^{911}\).
   3. Set base URL to referencing script's base URL\(^{911}\).

**Example**

referencingScriptOrModule is not usually null, but will be so for event handlers per the get the current value of the event handler\(^{946}\) algorithm. For example, given:

```html
<button onclick="import('./foo.mjs')">Click me</button>
```

If a click event occurs, then at the time the import() expression runs, GetActiveScriptOrModule will return null, which will be passed to this abstract operation when HostResolveImportedModule is called by FinishDynamicImport.

4. Let moduleMap be settings object's module map\(^{903}\).
5. Let url be the result of resolving a module specifier\(^{928}\) given base URL and specifier.
6. Assert: url is never failure, because resolving a module specifier\(^{928}\) must have been previously successful with these same two arguments (either while creating the corresponding module script\(^{920}\), or in fetch an import() module script graph\(^{915}\)).
7. Let resolved module script be moduleMap[url]. (This entry must exist for us to have gotten to this point.)
8. Assert: resolved module script is a module script\(^{911}\) (i.e., is not null or *fetching*).
9. Assert: resolved module script's record\(^{911}\) is not null.
10. Return resolved module script's record\(^{911}\).
8.1.6 Event loops

8.1.6.1 Definitions

To coordinate events, user interaction, scripts, rendering, networking, and so forth, user agents must use event loops as described in this section. Each agent has an associated event loop, which is unique to that agent.

The event loop of a similar-origin window agent is known as a window event loop. The event loop of a dedicated worker agent, shared worker agent, or service worker agent is known as a worker event loop. And the event loop of a worklet agent is known as a worklet event loop.

An event loop has one or more task queues. A task queue is a set of tasks.

Tasks encapsulate 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.

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.

Formally, a task is a struct which has:

Steps
- A series of steps specifying the work to be done by the task.

A source
- One of the task sources, used to group and serialize related tasks.

A document
- A Document associated with the task, or null for tasks that are not in a window event loop.
A script evaluation environment settings object set
A set of environment settings objects used for tracking script evaluation during the task.

A task is runnable if its document is either null or fully active.

Per its source field, each task is defined as coming from a specific task source. For each event loop, every task source must be associated with a specific task queue.

Note
Essentially, task sources are used within standards to separate logically-different types of tasks, which a user agent might wish to distinguish between. Task queues, are used by user agents to coalesce task sources within a given event loop.

Example
For example, a user agent could have one task queue for mouse and key events (to which the user interaction task source is associated), and another to which all other task sources are associated. Then, using the freedom granted in the initial step of the event loop processing model, it could give keyboard and mouse events preference over other tasks three-quarters of the time, keeping the interface responsive but not starving other task queues. Note that in this setup, the processing model still enforces that the user agent would never process events from any one task source out of order.

Each event loop has a currently running task, which is either a task or null. Initially, this is null. It is used to handle reentrancy.

Each event loop has a microtask queue, which is a queue of microtasks, initially empty. A microtask is a colloquial way of referring to a task that was created via the queue a microtask algorithm.

Each event loop has a performing a microtask checkpoint boolean, which is initially false. It is used to prevent reentrant invocation of the perform a microtask checkpoint algorithm.

8.1.6.2 Queuing tasks
To queue a task on a task source source, which performs a series of steps steps, optionally given an event loop event loop and a document document:

1. If event loop was not given, set event loop to the implied event loop.
2. If document was not given, set document to the implied document.
3. Let task be a new task.
4. Set task's steps to steps.
5. Set task's source to source.
7. Set task's script evaluation environment settings object set to an empty set.
8. Let queue be the task queue to which source is associated on event loop.
9. Append task to queue.

To queue a global task on a task source source, with a global object global and a series of steps steps:

1. Let event loop be global's relevant agent's event loop.
2. Let document be global's associated Document, if global is a Window object; otherwise null.
3. Queue a task given source, event loop, document, and steps.

To queue an element task on a task source source, with an element element and a series of steps steps:

1. Let global be element's relevant global object.
2. Queue a global task given source, global, and steps.

To queue a microtask which performs a series of steps steps, optionally given an event loop event loop and a document document:

1. If event loop was not given, set event loop to the implied event loop.
2. If document was not given, set document to the implied document.
3. Let microtask be a new task.
4. Set microtask's steps to steps.
5. Set microtask's source to the microtask task source.
7. Set task's script evaluation environment settings object set to an empty set.
8. Enqueue task on event loop's microtask queue.

The implied event loop when queuing a task is the one that can be deduced from the context of the calling algorithm. This is generally unambiguous, as most specification algorithms only ever involve a single agent (and thus a single event loop). The exception is algorithms involving or specifying cross-agent communication (e.g., between a window and a worker); for those cases, the implied event loop concept must not be relied upon and specifications must explicitly provide an event loop when queuing a task or microtask.

The implied document when queuing a task on an event loop event loop is determined as follows:

1. If event loop is not a window event loop, then return null.
2. If the task is being queued in the context of an element, then return the element's node document.
3. If the task is being queued in the context of a browsing context, then return the browsing context's active document.
4. If the task is being queued by or for a script, then return the script's settings object's responsible document.
5. Assert: this step is never reached, because one of the previous conditions must be true.

Both implied event loop and implied document are vaguely-defined and have a lot of action-at-a-distance. Perhaps we can come up with a more explicit architecture, while still avoiding all callers needing to explicitly specify the event loop and document.

8.1.6.3 Processing model

An event loop must continually run through the following steps for as long as it exists:

1. Let taskQueue be one of the event loop's task queues, chosen in an implementation-defined manner, with the constraint that the chosen task queue must contain at least one runnable task. If there is no such task queue, then jump to the microtasks step below.

   Remember that the microtask queue is not a task queue, so it will not be chosen in this step. However, a task queue to which the microtask task source is associated might be chosen in this step. In that case, the task chosen in the next step was originally a microtask, but it got moved as part of spinning the event loop.

2. Let oldestTask be the first runnable task in taskQueue, and remove it from taskQueue.
3. Set the event loop's currently running task to oldestTask.
4. Let `taskStartTime` be the current high resolution time.
5. Perform `oldestTask`'s steps.
6. Set the event loop's currently running task back to null.
7. Microtasks: Perform a microtask checkpoint.
8. Let `hasARenderingOpportunity` be false.
9. Let `now` be the current high resolution time [HRT].
10. Report the task's duration by performing the following steps:
    1. Let `top-level browsing contexts` be an empty set.
    2. For each environment settings object of `oldestTask`'s script evaluation environment settings object, append setting's top-level browsing context to `top-level browsing contexts`.
    3. Report long tasks, passing in `taskStartTime`, `now` (the end time of the task), `top-level browsing contexts`, and `oldestTask`.
11. Update the rendering: if this is a window event loop, then:
    1. Let `docs` be all Document objects whose relevant agent's event loop is this event loop, sorted arbitrarily except that the following conditions must be met:
       - Any Document B whose browsing context's container document is A must be listed after A in the list.
       - If there are two documents A and B whose browsing contexts are both child browsing contexts whose container documents are another Document C, then the order of A and B in the list must match the shadow-including tree order of their respective browsing context containers in C's node tree.
    2. Rendering opportunities: Remove from `docs` all Document objects whose browsing context do not have a rendering opportunity.
       - A browsing context has a rendering opportunity if the user agent is currently able to present the contents of the browsing context to the user, accounting for hardware refresh rate constraints and user agent throttling for performance reasons, but considering content presentable even if it's outside the viewport.
       - Browsing context rendering opportunities are determined based on hardware constraints such as display refresh rates and other factors such as page performance or whether the page is in the background. Rendering opportunities typically occur at regular intervals.
       - This specification does not mandate any particular model for selecting rendering opportunities. But for example, if the browser is attempting to achieve a 60Hz refresh rate, then rendering opportunities occur at a maximum of every 60th of a second (about 16.7ms). If the browser finds that a browsing context is not able to sustain this rate, it might drop to a more sustainable 30 rendering opportunities per second for that browsing context, rather than occasionally dropping frames. Similarly, if a browsing context is not visible, the user agent might decide to drop that page to a much slower 4 rendering opportunities per second, or even less.

    3. If `docs` is not empty, then set `hasARenderingOpportunity` to true.
4. Unnecessary rendering: Remove from `docs` all Document objects which meet both of the following conditions:
   - The user agent believes that updating the rendering of the Document's browsing context would have no visible effect, and
   - The Document's map of animation frame callbacks is empty.
5. Remove from `docs` all Document objects for which the user agent believes that it’s preferrable to skip updating the rendering for other reasons.

Note: This specification does not mandate any particular model for selecting rendering opportunities. But for example, if the browser is attempting to achieve a 60Hz refresh rate, then rendering opportunities occur at a maximum of every 60th of a second (about 16.7ms). If the browser finds that a browsing context is not able to sustain this rate, it might drop to a more sustainable 30 rendering opportunities per second for that browsing context, rather than occasionally dropping frames. Similarly, if a browsing context is not visible, the user agent might decide to drop that page to a much slower 4 rendering opportunities per second, or even less.
The step labeled Rendering opportunities prevents the user agent from updating the rendering when it is unable to present new content to the user (there's no rendering opportunity).

The step labeled Unnecessary rendering prevents the user agent from updating the rendering when there's no new content to draw.

This step enables the user agent to prevent the steps below from running for other reasons, for example, to ensure certain tasks are executed immediately after each other, with only microtask checkpoints interleaved (and without, e.g., animation frame callbacks interleaved). Concretely, a user agent might wish to coalesce timer callbacks together, with no intermediate rendering updates.

6. For each fully active Document in docs, flush autofocus candidates for that Document if its browsing context is a top-level browsing context.

7. For each fully active Document in docs, run the resize steps for that Document, passing in now as the timestamp. [CSSOMVIEW]

8. For each fully active Document in docs, run the scroll steps for that Document, passing in now as the timestamp. [CSSOMVIEW]

9. For each fully active Document in docs, evaluate media queries and report changes for that Document, passing in now as the timestamp. [CSSOMVIEW]

10. For each fully active Document in docs, update animations and send events for that Document, passing in now as the timestamp. [WEBANIMATIONS]

11. For each fully active Document in docs, run the fullscreen steps for that Document, passing in now as the timestamp. [FULLSCREEN]

12. For each fully active Document in docs, run the animation frame callbacks for that Document, passing in now as the timestamp.

13. For each fully active Document in docs, run the update intersection observations steps for that Document, passing in now as the timestamp. [INTERSECTIONOBSERVER]

14. Invoke the mark paint timing algorithm for each Document object in docs.

15. 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.

12. If all of the following are true

   1. this is a window event loop
   2. there is no task in this event loop's task queues whose document is fully active
   3. this event loop's microtask queue is empty
   4. hasARenderingOpportunity is false

   then for each Window object whose relevant agent's event loop is this event loop, run the start an idle period algorithm, passing the Window. [REQUESTIDLECALLBACK]

13. If this is a worker event loop, then:

   1. If this event loop's agent's single realm's global object is a supported DedicatedWorkerGlobalScope and the user agent believes that it would benefit from having its rendering updated at this time, then:
      1. Let now be the current high resolution time. [HRT]
      2. Run the animation frame callbacks for that DedicatedWorkerGlobalScope, passing in now as the timestamp.
      3. Update the rendering of that dedicated worker to reflect the current state.

Note

Similar to the notes for updating the rendering in a window event loop, a user agent can determine the rate of rendering in the dedicated worker.

2. If 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 described in the Web workers section below.

When a user agent is to perform a microtask checkpoint:

1. If the event loop's performing a microtask checkpoint is true, then return.
2. Set the event loop's performing a microtask checkpoint to true.
3. While the event loop's microtask queue is not empty:
   1. Let oldestMicrotask be the result of dequeuing from the event loop's microtask queue.
   2. Set the event loop's currently running task to oldestMicrotask.
   3. Run oldestMicrotask.

   **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.

4. Set the event loop's currently running task back to null.
5. For each environment settings object whose responsible event loop is this event loop, notify about rejected promises on that environment settings object.
6. Set the event loop's performing a microtask checkpoint to false.

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 ⌞.

Algorithm steps that say to spin the event loop until a condition goal is met are equivalent to substituting in the following algorithm steps:

1. Let task be the event loop's currently running task.

   **Note**
   task could be a microtask.

2. Let task source be task's source.
3. Let old stack be a copy of the JavaScript execution context stack.
4. Empty the JavaScript execution context stack.
5. Perform a microtask checkpoint.

   **Note**
   If task is a microtask this step will be a no-op due to performing a microtask checkpoint being true.

6. In parallel:
1. Wait until the condition goal is met.

2. Queue a task\textsuperscript{932} on task source to:
   
   1. Replace the JavaScript execution context stack with old stack.
   
   2. Perform any steps that appear after this spin the event loop\textsuperscript{936} instance in the original algorithm.

\begin{itemize}
\item[] \textbf{Note}
\end{itemize}

This resumes task.

7. Stop task, allowing whatever algorithm that invoked it to resume.

\begin{itemize}
\item[] \textbf{Note}
\end{itemize}

This causes the event loop\textsuperscript{931}’s main set of steps or the perform a microtask checkpoint\textsuperscript{936} algorithm to continue.

\begin{itemize}
\item[] \textbf{Note}
\end{itemize}

Unlike other algorithms in this and other specifications, which behave similar to programming-language function calls, spin the event loop\textsuperscript{936} is more like a macro, which saves typing and indentation at the usage site by expanding into a series of steps and operations.

\begin{itemize}
\item[] \textbf{Example}
\end{itemize}

An algorithm whose steps are:

1. Do something.
2. Spin the event loop\textsuperscript{936} until awesomeness happens.
3. Do something else.

is a shorthand which, after "macro expansion", becomes

1. Do something.
2. Let old stack be a copy of the JavaScript execution context stack.
3. Empty the JavaScript execution context stack.
4. Perform a microtask checkpoint\textsuperscript{936}.
5. \textbf{In parallel}\textsuperscript{942}:
   
   1. Wait until awesomeness happens.
   
   2. Queue a task\textsuperscript{932} on the task source in which "do something" was done to:
      
      1. Replace the JavaScript execution context stack with old stack.
      
      2. Do something else.

\begin{itemize}
\item[] \textbf{Example}
\end{itemize}

Here is a more full example of the substitution, where the event loop is spun from inside a task that is queued from work in parallel. The version using spin the event loop\textsuperscript{936}:

1. \textbf{In parallel}\textsuperscript{942}:
   
   1. Do parallel thing 1.
   
   2. Queue a task\textsuperscript{932} on the DOM manipulation task source\textsuperscript{938} to:
      
      1. Do task thing 1.
      
      2. Spin the event loop\textsuperscript{936} until awesomeness happens.
Some of the algorithms in this specification, for historical reasons, require the user agent to **pause** while running a **task** until a **condition** is met. This means running the following steps:

1. **In parallel**:
   1. Do parallel thing 1.
   2. Let *old stack* be null.
   3. **Queue a task** on the DOM manipulation task source to:
      1. Do task thing 1.
      2. Set *old stack* to a copy of the JavaScript execution context stack.
      3. Empty the JavaScript execution context stack.
      4. **Perform a microtask checkpoint**.

2. Wait until awesomeness happens.

3. **Queue a task** on the DOM manipulation task source to:
   1. Replace the JavaScript execution context stack with *old stack*.
   2. Do task thing 2.

4. Do parallel thing 2.

The fully expanded version:

1. **In parallel**:
   1. Do parallel thing 1.
   2. Let *old stack* be null.
   3. **Queue a task** on the DOM manipulation task source to:
      1. Do task thing 1.
      2. Set *old stack* to a copy of the JavaScript execution context stack.
      3. Empty the JavaScript execution context stack.
      4. **Perform a microtask checkpoint**.

4. Wait until awesomeness happens.

5. **Queue a task** on the DOM manipulation task source to:
   1. Replace the JavaScript execution context stack with *old stack*.
   2. Do task thing 2.

6. Do parallel thing 2.

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.

⚠ **Warning!**

**Pausing** is highly detrimental to the user experience, especially in scenarios where a single event loop is shared among multiple documents. User agents are encouraged to experiment with alternatives to **pausing**, such as spinning the event loop or even simply proceeding without any kind of suspended execution at all, insofar as it is possible to do so while preserving compatibility with existing content. This specification will happily change if a less-drastic alternative is discovered to be web-compatible.

In the interim, implementers should be aware that the variety of alternatives that user agents might experiment with can change subtle aspects of event loop behavior, including task and microtask timing. Implementations should continue experimenting even if doing so causes them to violate the exact semantics implied by the **pause** operation.

### 8.1.6.4 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[^932] 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[^931] queued[^932] with the user interaction task source[^939] [UIEVENTS][^1277].

The networking task source

This task source[^932] is used for features that trigger in response to network activity.

The history traversal task source

This task source[^932] is used to queue calls to history.back()[^861] and similar APIs.

8.1.6.5 Dealing with the event loop from other specifications[^93]

Writing specifications that correctly interact with the event loop[^931] can be tricky. This is compounded by how this specification uses concurrency-model-independent terminology, so we say things like “event loop[^931]” and “in parallel[^42]” instead of using more familiar model-specific terms like "main thread" or "on a background thread".

By default, specification text generally runs on the event loop[^931]. This falls out from the formal event loop processing model[^933], in that you can eventually trace most algorithms back to a task[^931] queued[^932] there.

Example

The algorithm steps for any JavaScript method will be invoked by author code calling that method. And author code can only be run via queued tasks, usually originating somewhere in the script processing model[^619].

From this starting point, the overriding guideline is that any work a specification needs to perform that would otherwise block the event loop[^931] must instead be performed in parallel[^42] with it. This includes (but is not limited to):

- performing heavy computation;
- displaying a user-facing prompt;
- performing operations which could require involving outside systems (i.e. "going out of process").

The next complication is that, in algorithm sections that are in parallel[^42], you must not create or manipulate objects associated to a specific JavaScript realm, global[^904], or environment settings object[^903]. (Stated in more familiar terms, you must not directly access main-thread artifacts from a background thread.) Doing so would create data races observable to JavaScript code, since after all, your algorithm steps are running in parallel[^42] to the JavaScript code.

You can, however, manipulate specification-level data structures and values from Infra, as those are realm-agnostic. They are never directly exposed to JavaScript without a specific conversion taking place (often via Web IDL). [INFRA][^1277] [WEBIDL][^1277]

To affect the world of observable JavaScript objects, then, you must queue a task[^932] to perform any such manipulations. This ensures your steps are properly interleaved with respect to other things happening on the event loop[^931]. Furthermore, you must choose a task source[^932] when queuing a task[^932]; this governs the relative order of your steps versus others. If you are unsure which task source[^932] to use, pick one of the generic task sources[^938] that sounds most applicable.

Most invocations of queue a task[^932] use the implied event loop[^933], i.e., the one that is obvious from context. That is because it is very rare for algorithms to be invoked in contexts involving multiple event loops. (Unlike contexts involving multiple global objects, which happen all the time!) So unless you are writing a specification which, e.g., deals with manipulating workers[^911], you can omit this argument when queueing a task[^932].

Putting this all together, we can provide a template for a typical algorithm that needs to do work asynchronously:

1. Do any synchronous setup work, while still on the event loop[^931]. This may include converting realm-specific JavaScript values into realm-agnostic specification-level values.
2. Perform a set of potentially-expensive steps in parallel[^42], operating entirely on realm-agnostic values, and producing a realm-agnostic result.
3. Queue a task[^932], on a specified task source[^932], to convert the realm-agnostic result back into observable effects on the observable world of JavaScript objects on the event loop[^931].
The following is an algorithm that "encrypts" a passed-in list of scalar value strings input, after parsing them as URLs:

1. Let urls be an empty list.
2. For each string of input:
   1. Let parsed be the result of parsing string relative to the current settings object.
   2. If parsed is failure, return a promise rejected with a "SyntaxError" DOMException.
   3. Let serialized be the result of applying the URL serializer to parsed.
   4. Append serialized to urls.
3. Let realm be the current Realm Record.
4. Let p be a new promise.
5. Run the following steps in parallel:
   1. Let encryptedURLs be an empty list.
   2. For each url of urls:
      1. Wait 100 milliseconds, so that people think we're doing heavy-duty encryption.
      2. Let encrypted be a new string derived from url, whose nth code unit is equal to url's nth code unit plus 13.
      3. Append encrypted to encryptedURLs.
   3. Queue a task, on the networking task source, to perform the following steps:
      1. Let array be the result of converting encryptedURLs to a JavaScript array, in realm.
      2. Resolve p with array.

Here are several things to notice about this algorithm:

- It does its URL parsing up front, on the event loop, before going to the in parallel steps. This is necessary, since parsing depends on the current settings object, which would no longer be current after going in parallel.
- Alternately, it could have saved a reference to the current settings object's API base URL and used it during the in parallel steps; that would have been equivalent. However, we recommend instead doing as much work as possible up front, as this example does. Attempting to save the correct values can be error prone; for example, if we'd saved just the current settings object, instead of its API base URL, there would have been a potential race.
- It implicitly passes a list of strings from the initial steps to the in parallel steps. This is OK, as both lists and strings are realm-agnostic.
- It performs "expensive computation" (waiting for 100 milliseconds per input URL) during the in parallel steps, thus not blocking the main event loop.
- Promises, as observable JavaScript objects, are never created and manipulated during the in parallel steps; p is created before entering those steps, and then is manipulated during a task that is queued specifically for that purpose.
- The creation of a JavaScript array object also happens during the queued task, and is careful to specify which realm it creates the array in since is no longer obvious from context.

(On these last two points, see also w3ctag/promises-guide issue #52, heycam/webidl issue #135, and heycam/webidl issue #371, where we are still mulling over the subtleties of the above promise-resolution pattern.)

Another thing to note is that, in the event this algorithm was called from a Web IDL-specified operation taking a sequence input, there was an automatic conversion from realm-specific JavaScript objects provided by the author as input, into the realm-agnostic sequence Web IDL type, which we then treat as a list of scalar value strings. So depending on how your specification is structured, there may be other implicit steps happening on the main event loop that play a part in this...
8.1.7 Events

8.1.7.1 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 is a struct with two items:

- a value, which is either null, a callback object, or 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.

- a listener, which is either null or an event listener responsible for running the event handler processing algorithm. Initially, an event handler’s listener must be set to null.

Event handlers are exposed in 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.

For both of these two ways, the event handler is exposed through a name, which is a string that always starts with "on" and is followed by the name of the event for which the handler is intended.

Most of the time, the object that exposes an event handler is the same as the object on which the corresponding event listener is added. However, the body and frameset elements expose several event handlers that act upon the element’s Window object, if one exists. In either case, we call the object an event handler acts upon the target of that event handler.

To determine the target of an event handler, given an EventTarget object eventTarget on which the event handler is exposed, and an event handler name, the following steps are taken:

1. If eventTarget is not a body element or a frameset element, then return eventTarget.
2. If name is not the name of an attribute member of the WindowEventHandlers interface mixin and the Window-reflecting body element event handler set does not contain name, then return eventTarget.
3. If eventTarget’s node document is not an active document, then return null.
4. Return eventTarget’s node document’s relevant global object.

Each EventTarget object that has one or more event handlers specified has an associated event handler map, which is a map of strings representing names of event handlers to event handlers.

When an EventTarget object that has one or more event handlers specified is created, its event handler map must be initialized such that it contains an entry for each event handler that has that object as target, with items in those event handlers set to their initial values.
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**.

The getter of an **event handler IDL attribute** with name **name**, when called, must run these steps:

1. Let `eventTarget` be the result of determining the target of an **event handler** given this object and **name**.
2. If `eventTarget` is null, then return null.
3. Return the result of getting the current value of the **event handler** given `eventTarget` and **name**.

The setter of an **event handler IDL attribute** with name **name**, when called, must run these steps:

1. Let `eventTarget` be the result of determining the target of an **event handler** given this object and **name**.
2. If `eventTarget` is null, then return.
3. If the given value is null, then deactivate an **event handler** given `eventTarget` and **name**.
4. Otherwise:
   1. Let `handlerMap` be `eventTarget`'s **event handler map**.
   2. Let `eventHandler` be `handlerMap[**name**]`.
   3. Set `eventHandler`'s **value** to the given value.
   4. Activate an **event handler** given `eventTarget` and **name**.

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.

The following **attribute change steps** are used to synchronize between **event handler content attributes** and **event handlers**: [DOM](#).

1. If `namespace` is not null, or `localName` is not the name of an **event handler content attribute** on `element`, then return.
2. Let `eventTarget` be the result of determining the target of an **event handler** given `element` and `localName`.
3. If `eventTarget` is null, then return.
4. If `value` is null, then deactivate an **event handler** given `eventTarget` and `localName`.
5. Otherwise:
   1. If the Should element's inline behavior be blocked by Content Security Policy? algorithm returns "Blocked" when executed upon `element`, "script attribute", and `value`, then return. [CSP](#)
   2. Let `handlerMap` be `eventTarget`'s **event handler map**.
3. Let `eventHandler` be `handlerMap[localName]`.

4. Let `location` be the script location that triggered the execution of these steps.

5. Set `eventHandler`'s `value` to the internal raw uncompiled handler value/`location`.

6. **Activate an event handler** given `eventTarget` and `localName`.

**Note**

*Per the DOM Standard, these steps are run even if oldValue and value are identical (setting an attribute to its current value), but not if oldValue and value are both null (removing an attribute that doesn't currently exist). [DOM]*

To **deactivate an event handler** given an `EventTarget` object `eventTarget` and a string `name` that is the `name` of an event handler, run these steps:

1. Let `handlerMap` be `eventTarget`'s event handler map.
2. Let `eventHandler` be `handlerMap[name]`.
3. Set `eventHandler`'s `value` to null.
4. Let `listener` be `eventHandler`'s `listener`.
5. If `listener` is not null, then remove an event listener with `eventTarget` and `listener`.
6. Set `eventHandler`'s `listener` to null.

To **erase all event listeners and handlers** given an `EventTarget` object `eventTarget`, run these steps:

1. If `eventTarget` has an associated event handler map, then for each `name` → `eventHandler` of `eventTarget`'s associated event handler map, deactivate an event handler given `eventTarget` and `name`.
2. Remove all event listeners given `eventTarget`.

**Note**

This algorithm is used to define `document.open()`.

To **activate an event handler** given an `EventTarget` object `eventTarget` and a string `name` that is the `name` of an event handler, run these steps:

1. Let `handlerMap` be `eventTarget`'s event handler map.
2. Let `eventHandler` be `handlerMap[name]`.
3. If `eventHandler`'s `listener` is not null, then return.
4. Let `callback` be the result of creating a Web IDL `EventListener` instance representing a reference to a function of one argument that executes the steps of the event handler processing algorithm, given `eventTarget`, `name`, and its argument.

The `EventListener`'s `callback context` can be arbitrary; it does not impact the steps of the event handler processing algorithm. [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 code, and processing the code's return value.

5. Let `listener` be a new `event listener` whose `type` is the event handler event type corresponding to `eventHandler` and `callback` is `callback`.

**Note**

To be clear, an event listener is different from an `EventListener`.

6. Add an event listener with `eventTarget` and `listener`. 
7. Set `eventHandler's listener` to `listener`.

**Note**

The event listener registration happens only if the `event handler`'s `value` is being set to non-null, and the `event handler` is not already activated. Since listeners are called in the order they were registered, assuming no deactivation occurred, the order of event listeners for a particular event type will always be:

1. the event listeners registered with `addEventListener()` before the first time the `event handler`'s `value` was set to non-null
2. then the callback to which it is currently set, if any
3. and finally the event listeners registered with `addEventListener()` after the first time the `event handler`'s `value` was set to non-null.

**Example**

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.

```html
<button id="test">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') }; // event handler is activated here
button.addEventListener('click', function () { alert('FOUR') }, false);
</script>
```

However, in the following example, the event handler is deactivated after its initial activation (and its event listener is removed), before being reactivated at a later time. The page will show five alerts with "ONE", "TWO", "THREE", "FOUR", and "FIVE" respectively, in order.

```html
<button id="test">Start Demo</button>

<script>
var button = document.getElementById('test');
button.addEventListener('click', function () { alert('ONE') }, false);
button.setAttribute('onclick', "alert('NOT CALLED')"); // event handler is activated here
button.addEventListener('click', function () { alert('TWO') }, false);
button.onclick = null; // but deactivated here
button.addEventListener('click', function () { alert('THREE') }, false);
button.onclick = function () { alert('FOUR') }; // and re-activated here
button.addEventListener('click', function () { alert('FIVE') }, 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 `EventTarget` object `eventTarget`, a string `name` representing the `name` of an `event handler`, and an `Event` object event is as follows:

1. Let `callback` be the result of getting the current value of the `event handler` given `eventTarget` and `name`.
2. If `callback` is null, then return.
3. Let `special error event handling` be true if `event` is an `ErrorEvent` object, `event`'s `type` is `error`, and `event`'s `currentTarget` implements the `WindowOrWorkerGlobalScope` mixin. Otherwise, let `special error event handling` be false.
4. Process the `Event` object event as follows:
If special error event handling is true
Invoke callback with five arguments, the first one having the value of event's `message` attribute, the second having the value of event's `filename` attribute, the third having the value of event's `linenumber` attribute, the fourth having the value of event's `columnnumber` attribute, the fifth having the value of event's `error` attribute, and with the `callback this value` set to event'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 `event`, with the `callback this value` set to `event's currentTarget`. Let `return value` be the callback's return value. [WEBIDL]

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.)

5. Process `return value` as follows:

- If `event` is a `BeforeUnloadEvent` object and `event's type` is `beforeunload`:

  **Note**
  In this case, the `event handler IDL attribute`'s type will be `OnBeforeUnloadEventHandler`, so `return value` will have been coerced into either null or a `DOMString`.

  If `return value` is not null, then:
  1. Set `event's canceled flag`.
  2. If `event's returnValue` attribute's value is the empty string, then set `event's returnValue` attribute's value to `return value`.

- If special error event handling is true
  If `return value` is true, then set `event's canceled flag`.

- Otherwise
  If `return value` is false, then set `event's canceled flag`.

  **Note**
  If we've gotten to this "Otherwise" clause because `event's type` is `beforeunload` but `event` is not a `BeforeUnloadEvent` object, then `return value` will never be false, since in such cases `return value` will have been coerced into either null or a `DOMString`.

The `EventHandler` callback function type represents a callback used for event handlers. It is represented in Web IDL as follows:

```idl
[LegacyTreatNonObjectAsNull]
callback EventHandlerNonNull = any (Event event);
typedef EventHandlerNonNull? EventHandler;
```

**Note**
In JavaScript, any `Function` object implements this interface.

**Example**
For example, the following document fragment:

```html
<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.

There are two exceptions in the platform, for historical reasons:

- The `onerror` handlers on global objects, where returning true cancels the event
- The `onbeforeunload` handler, where returning any non-null and non-undefined value will cancel the event.

For historical reasons, the `onerror` handler has different arguments:

```idl
[LegacyTreatNonObjectAsNull]
callback OnErrorEventHandlerNonNull = any ((Event or DOMString) event, optional DOMString source, optional unsigned long lineno, optional unsigned long colno, optional any error);
typedef OnErrorEventHandlerNonNull? OnErrorEventHandler;
```

```example
window.onerror = (message, source, lineno, colno, error) => { ... };
```

Similarly, the `onbeforeunload` handler has a different return value:

```idl
[LegacyTreatNonObjectAsNull]
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 given an `EventTarget` object `eventTarget` and a string `name` that is the `name` of an event handler, it must run these steps:

1. Let `handlerMap` be `eventTarget`'s `event handler map`.
2. Let `eventHandler` be `handlerMap`[`name`].
3. If `eventHandler`'s `value` is an **internal raw uncompiled handler**, then:
   1. If `eventTarget` is an element, then let `element` be `eventTarget`, and `document` be `element`'s `node document`. Otherwise, `eventTarget` is a `Window` object, let `element` be null, and `document` be `eventTarget`'s associated `Document`.
   2. If `scripting is disabled` for `document`, then return null.
   3. Let `body` be the uncompiled script body in `eventHandler`'s `value`.
   4. Let `location` be the location where the script body originated, as given by `eventHandler`'s `value`.
   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 `settings object` be the `relevant settings object` of `document`.
   7. If `body` is not parsable as `FunctionBody` or if parsing detects an `early error`, then follow these substeps:
      1. Set `eventHandler`'s `value` to null.

Note

This does not `deactivate` the event handler, which additionally removes the event handler's listener (if present).
2. **Report the error** for the appropriate **script** and with the appropriate position (line number and column number) given by **location**, using **settings object's** **global object**. If the error is still **not handled** after this, then the error may be reported to a developer console.

3. Return null.

8. Push **settings object's** **realm execution context** onto the **JavaScript execution context stack**: it is now the running **JavaScript execution context**.

   **Note**  
   *This is necessary so the subsequent invocation of **OrdinaryFunctionCreate** takes place in the correct **JavaScript Realm***.

9. Let **function** be the result of calling **OrdinaryFunctionCreate**, with arguments:

   **functionPrototype**
   
   %Function.prototype%

   **sourceText**

   → If **name** is **onerror** and **eventTarget** is a **Window** object
   
   The string formed by concatenating "function ", **name**, "(event, source, lineno, colno, error) (", U+000A LF, body, U+000A LF, and ")".
   
   → Otherwise
   
   The string formed by concatenating "function ", **name**, "(event) (", U+000A LF, body, U+000A LF, and ")".

   **ParameterList**

   → If **name** is **onerror** and **eventTarget** is 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.

   **thisMode**

   non-lexical-this

   **Scope**

   1. Let **realm** be **settings object's** **Realm**.
   
   2. Let **scope** be **realm**.\([\text{[GlobalEnv]}]\).
   
   3. If **eventHandler** is an element's **event handler**, then set **scope** to **NewObjectEnvironment**(document, **scope**).
   
   (Otherwise, **eventHandler** is a **Window** object's **event handler**.)
   
   4. If **form owner** is not null, then set **scope** to **NewObjectEnvironment**(form owner, **scope**).
   
   5. If **element** is not null, then set **scope** to **NewObjectEnvironment**(element, **scope**).
   
   6. Return **scope**.

10. Remove **settings object's** **realm execution context** from the **JavaScript execution context stack**.

11. Set **function**.\([\text{[ScriptOrModule]}]\) to null.

   **Note**  
   *This is done because the default behavior, of associating the created function with the nearest **script** on the stack, can lead to path-dependent results. For example, an event handler which is first invoked by user interaction would end up with null \([\text{[ScriptOrModule]}]\) (since then this algorithm would be first invoked when the **active script** is null), whereas one that is first invoked by dispatching an event from script would have its \([\text{[ScriptOrModule]}]\) set to that script.*
Instead, we just always set [[ScriptOrModule]] to null. This is more intuitive anyway; the idea that the first script which dispatches an event is somehow responsible for the event handler code is dubious.

In practice, this only affects the resolution of relative URLs via `import()`, which consult the base URL of the associated script. Nulling out [[ScriptOrModule]] means that `HostResolveImportedModule` and `HostImportModuleDynamically` will fall back to the current settings object's API base URL.

12. Set `eventHandler`'s `value` to the result of creating a Web IDL `EventHandler` callback function object whose object reference is `function` and whose callback context is `settings object`.

4. Return `eventHandler`'s `value`.

8.1.7.2 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:

<table>
<thead>
<tr>
<th>Event handler</th>
<th>Event handler event type</th>
</tr>
</thead>
<tbody>
<tr>
<td>onabort</td>
<td>abort</td>
</tr>
<tr>
<td>onauxclick</td>
<td>auxclick</td>
</tr>
<tr>
<td>oncancel</td>
<td>cancel</td>
</tr>
<tr>
<td>oncanplay</td>
<td>canplay</td>
</tr>
<tr>
<td>oncanplaythrough</td>
<td>canplaythrough</td>
</tr>
<tr>
<td>onchange</td>
<td>change</td>
</tr>
<tr>
<td>onclic</td>
<td>click</td>
</tr>
<tr>
<td>onclose</td>
<td>close</td>
</tr>
<tr>
<td>oncontextmenu</td>
<td>contextmenu</td>
</tr>
<tr>
<td>oncuechange</td>
<td>cuechange</td>
</tr>
<tr>
<td>ondblclick</td>
<td>dblclick</td>
</tr>
<tr>
<td>ondrag</td>
<td>drag</td>
</tr>
<tr>
<td>ondragend</td>
<td>dragend</td>
</tr>
<tr>
<td>ondragenter</td>
<td>dragenter</td>
</tr>
<tr>
<td>ondragleave</td>
<td>dragleave</td>
</tr>
<tr>
<td>ondragover</td>
<td>dragover</td>
</tr>
<tr>
<td>ondragstart</td>
<td>dragstart</td>
</tr>
<tr>
<td>ondrop</td>
<td>drop</td>
</tr>
<tr>
<td>ondurationchange</td>
<td>durationchange</td>
</tr>
<tr>
<td>onemptied</td>
<td>emptied</td>
</tr>
<tr>
<td>onended</td>
<td>ended</td>
</tr>
<tr>
<td>onformdata</td>
<td>formdata</td>
</tr>
<tr>
<td>oninput</td>
<td>input</td>
</tr>
<tr>
<td>oninvalid</td>
<td>invalid</td>
</tr>
<tr>
<td>onkeydown</td>
<td>keydown</td>
</tr>
<tr>
<td>onkeypress</td>
<td>keypress</td>
</tr>
<tr>
<td>onkeyup</td>
<td>keyup</td>
</tr>
<tr>
<td>onoleddata</td>
<td>loadeddata</td>
</tr>
<tr>
<td>onoleddatedata</td>
<td>loadeddata</td>
</tr>
<tr>
<td>onoleddstart</td>
<td>loadstart</td>
</tr>
<tr>
<td>onoleedmouse</td>
<td>mousedown</td>
</tr>
<tr>
<td>onoleemouseenter</td>
<td>mouseenter</td>
</tr>
<tr>
<td>onoleemouseleave</td>
<td>mouseleave</td>
</tr>
<tr>
<td>onoleemovenone</td>
<td>mousenone</td>
</tr>
<tr>
<td>onoleemouseout</td>
<td>mouseout</td>
</tr>
<tr>
<td>onoleemouseover</td>
<td>mouseover</td>
</tr>
</tbody>
</table>
The following are the event handlers\(^{941}\) (and their corresponding event handler event types\(^{943}\)) that must be supported by all HTML elements\(^{944}\); other than body\(^{179}\) and frameset\(^{1224}\) elements, as both event handler content attributes\(^{942}\) and event handler IDL attributes\(^{942}\); that must be supported by all Document\(^{913}\) objects, as event handler IDL attributes\(^{942}\); and that must be supported by all Window\(^{827}\) objects, as event handler content attributes\(^{942}\) and event handler IDL attributes\(^{942}\) on the Window\(^{827}\) objects themselves, and with corresponding event handler content attributes\(^{942}\) and event handler IDL attributes\(^{942}\) exposed on all body\(^{979}\) and frameset\(^{1224}\) elements that are owned by that Window\(^{827}\) object's associated Document\(^{828}\):

<table>
<thead>
<tr>
<th>Event handler(^{941})</th>
<th>Event handler event type(^{943})</th>
</tr>
</thead>
<tbody>
<tr>
<td>onblur</td>
<td>blur(^{926})</td>
</tr>
<tr>
<td>onerror</td>
<td>error(^{926})</td>
</tr>
<tr>
<td>onfocus</td>
<td>focus(^{926})</td>
</tr>
<tr>
<td>onload</td>
<td>load(^{1296})</td>
</tr>
<tr>
<td>onresize</td>
<td>resize</td>
</tr>
<tr>
<td>onscroll</td>
<td>scroll</td>
</tr>
<tr>
<td>onafterprint</td>
<td>afterprint(^{926})</td>
</tr>
<tr>
<td>onbeforeprint</td>
<td>beforeprint(^{1292})</td>
</tr>
<tr>
<td>onbeforeunload</td>
<td>beforeunload(^{1296})</td>
</tr>
<tr>
<td>onhashchange</td>
<td>hashchange(^{926})</td>
</tr>
<tr>
<td>onlanguagechange</td>
<td>languagechange(^{1256})</td>
</tr>
</tbody>
</table>

We call the set of the names\(^{941}\) of the event handlers\(^{941}\), listed in the first column of this table the **Window-reflecting body element event handler set**.

The following are the event handlers\(^{941}\) (and their corresponding event handler event types\(^{943}\)) that must be supported by Window\(^{827}\) objects, as event handler IDL attributes\(^{942}\) on the Window\(^{827}\) objects themselves, and with corresponding event handler content attributes\(^{942}\) and event handler IDL attributes\(^{942}\) exposed on all body\(^{979}\) and frameset\(^{1224}\) elements that are owned by that Window\(^{827}\) object's associated Document\(^{828}\):

<table>
<thead>
<tr>
<th>Event handler(^{941})</th>
<th>Event handler event type(^{943})</th>
</tr>
</thead>
<tbody>
<tr>
<td>onafterprint</td>
<td>afterprint(^{926})</td>
</tr>
<tr>
<td>onbeforeprint</td>
<td>beforeprint(^{1292})</td>
</tr>
<tr>
<td>onbeforeunload</td>
<td>beforeunload(^{1296})</td>
</tr>
<tr>
<td>onhashchange</td>
<td>hashchange(^{926})</td>
</tr>
<tr>
<td>onlanguagechange</td>
<td>languagechange(^{1256})</td>
</tr>
</tbody>
</table>
This list of event handlers is reified as event handler IDL attributes through the WindowEventHandlers interface mixin.

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:

<table>
<thead>
<tr>
<th>Event handler</th>
<th>Event handler event type</th>
</tr>
</thead>
<tbody>
<tr>
<td>onmessage</td>
<td>message</td>
</tr>
<tr>
<td>onmessageerror</td>
<td>messageerror</td>
</tr>
<tr>
<td>onoffline</td>
<td>offline</td>
</tr>
<tr>
<td>ononline</td>
<td>online</td>
</tr>
<tr>
<td>onpagehide</td>
<td>pagehide</td>
</tr>
<tr>
<td>onpageshow</td>
<td>pageshow</td>
</tr>
<tr>
<td>onpopstate</td>
<td>popstate</td>
</tr>
<tr>
<td>onrejectionhandled</td>
<td>rejectionhandled</td>
</tr>
<tr>
<td>onstorage</td>
<td>storage</td>
</tr>
<tr>
<td>onunhandledrejection</td>
<td>unhandledrejection</td>
</tr>
<tr>
<td>onunload</td>
<td>unload</td>
</tr>
</tbody>
</table>

This list of event handlers is reified as event handler IDL attributes through the DocumentAndElementEventHandlers interface mixin.

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:

<table>
<thead>
<tr>
<th>Event handler</th>
<th>Event handler event type</th>
</tr>
</thead>
<tbody>
<tr>
<td>onreadystatechange</td>
<td>readystatechange</td>
</tr>
</tbody>
</table>

8.1.7.2.1 IDL definitions

```idl
interface mixin GlobalEventHandlers {
  attribute EventHandler onabort;
  attribute EventHandler onauxclick;
  attribute EventHandler onblur;
  attribute EventHandler oncancel;
  attribute EventHandler oncancelayplay;
  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 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 onformdata;
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 onmousedownload;
[LegacyLenientThis] attribute EventHandler onmouseenter;
[LegacyLenientThis] attribute EventHandler onmouseleave;
attribute EventHandler onmousemove;
attribute EventHandler onmouseout;
attribute EventHandler onmouseover;
attribute EventHandler onmouseup;
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 onsecuritypolicyviolation;
attribute EventHandler onseeked;
attribute EventHandler onseeking;
attribute EventHandler onselect;
attribute EventHandler onslotchange;
attribute EventHandler onstalled;
attribute EventHandler onsubmit;
attribute EventHandler onsuspend;
attribute EventHandler ontimeupdate;
attribute EventHandler ontoggle;
attribute EventHandler onvolumechange;
attribute EventHandler onwaiting;
attribute EventHandler onwebkitanimationend;
attribute EventHandler onwebkitanimationiteration;
attribute EventHandler onwebkitanimationstart;
attribute EventHandler onwebkittransitionend;
attribute EventHandler onwheel;
};

interface mixin WindowEventHandlers {
    attribute EventHandler onafterprint;
    attribute EventHandler onbeforeprint;
    attribute OnBeforeUnloadEventHandler onbeforeunload;
    attribute EventHandler onhashchange;
    attribute EventHandler onlanguagechange;
    attribute EventHandler onmessage;
    attribute EventHandler onmessageerror;
    attribute EventHandler onoffline;
    attribute EventHandler ononline;
    attribute EventHandler onpagehide;
    attribute EventHandler onpageshow;
}
### 8.1.7.3 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 synthetic pointer event named `e` at `target`, with an optional not trusted flag, means running these steps:

1. Let `event` be the result of creating an event using `PointerEvent`.
2. Initialize `event`'s `type` attribute to `e`.
3. Initialize `event`'s `bubbles` and `cancelable` attributes to true.
4. Set `event`'s `composed` flag.
5. If the `not trusted` flag is set, initialize `event`'s `isTrusted` attribute to false.
6. Initialize `event`'s `ctrlKey`, `shiftKey`, `altKey`, and `metaKey` attributes according to the current state of the key input device, if any (false for any keys that are not available).
7. Initialize `event`'s `view` attribute to `target`'s `node document`'s `Window` object, if any, and null otherwise.
8. `event`'s `getModifierState()` method is to return values appropriately describing the current state of the key input device.
9. Return the result of dispatching `event` at `target`.

**Firing a click** event at `target` means firing a synthetic pointer event named `click` at `target`.

### 8.2 The `WindowOrWorkerGlobalScope` mixin

The `WindowOrWorkerGlobalScope` mixin is for use of APIs that are to be exposed on `Window` and `WorkerGlobalScope` objects.

**Note**

Other standards are encouraged to further extend it using partial interface mixin `WindowOrWorkerGlobalScope` { ... } along with an appropriate reference.
The `btoa()` and `atob()` methods allow developers to transform content to and from the base64 encoding.

8.3 Base64 utility methods

The `atob()` and `btoa()` methods allow developers to transform content to and from the base64 encoding.
The `btoa(data)` method must throw an "InvalidCharacterError" DOMException if `data` contains any character whose code point is greater than U+00FF. Otherwise, the user agent must convert `data` to a byte sequence whose `n`th byte is the eight-bit representation of the `n`th code point of `data`, and then must apply forgiving-base64 encode to that byte sequence and return the result.

The `atob(data)` method, when invoked, must run the following steps:

1. Let `decodedData` be the result of running forgiving-base64 decode on `data`.
2. If `decodedData` is failure, then throw an "InvalidCharacterError" DOMException.
3. Return `decodedData`.

### 8.4 Dynamic markup insertion

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 XML documents (and the XML parser).

Document objects have a throw-on-dynamic-markup-insertion counter, which is used in conjunction with the create an element for the token algorithm to prevent custom element constructors from being able to use `document.open()`, `document.close()`, and `document.write()` when they are invoked by the parser. Initially, the counter must be set to zero.

### 8.4.1 Opening the input stream

For web developers (non-normative)

`document = document . open( )`

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.

The resulting Document has an HTML parser associated with it, which can be given data to parse using `document.write()`.

The method has no effect if the Document is still being parsed.

Throws an "InvalidStateError" DOMException if the Document is an XML document.

Throws an "InvalidStateError" DOMException if the parser is currently executing a custom element constructor.

`window = document . open( url, name, features )`

Works like the window.open() method.

Document objects have an ignore-opens-during-unload counter, which is used to prevent scripts from invoking the...
The **document open** steps, given a **document**, are as follows:

1. If **document** is an **XML document**, then throw an "InvalidStateError" **DOMException** exception.
2. If **document**'s **throw-on-dynamic-markup-insertion** counter is greater than 0, then throw an "InvalidStateError" **DOMException**.
3. Let entryDocument be the **entry global object**'s associated **Document**.
4. If **document**'s **origin** is not **same origin** to entryDocument's **origin**, then throw a "SecurityError" **DOMException**.
5. If **document** has an **active parser** whose **script nesting level** is greater than 0, then return **document**.
6. Similarly, if **document**'s **ignore-opens-during-unload** counter is greater than 0, then return **document**.
7. If **document**'s **active parser was aborted** is true, then return **document**.
8. If **document**'s **browsing context** is non-null and there is an existing attempt to **navigate** **document**'s **browsing context**, then **stop document loading** given **document**.
9. For each **shadow-including inclusive descendant** node of **document**, erase all event listeners and handlers given **node**.
10. If **document** is the **associated Document** of **document**'s **relevant global object**, then erase all event listeners and handlers given **document**'s **relevant global object**.
11. Replace all with null within **document**, without firing any mutation events.
12. If **document** is **fully active**, then:
   1. Let newURL be a copy of entryDocument's **URL**.
   2. If entryDocument is not **document**, then set newURL's **fragment** to null.
   3. Run the **URL and history update steps** with **document** and newURL.
13. If **document**'s **iframe load in progress** flag is set, then set **document**'s **mute iframe load** flag.
14. Set **document** to **no-quirks mode**.
15. Create a new **HTML parser** and associate it with **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.
16. Set the current document readiness of document to “loading”.

17. Finally, set the insertion point to point at just before the end of the input stream (which at this point will be empty).


**Note**

The document open steps do not affect whether a Document is ready for post-load tasks or completely loaded.

The open method must return the result of running the document open steps with this Document object.

**Note**

The unused1 and unused2 arguments are ignored, but kept in the IDL to allow code that calls the function with one or two arguments to continue working. They are necessary due to Web IDL overload resolution algorithm rules, which would throw a TypeError exception for such calls had the arguments not been there. [WEBIDL]

The open method must run these steps:

1. If this Document object is not an active document, then throw an "InvalidAccessError" DOMException exception.
2. Return the result of running the window open steps with url, name, and features.

### 8.4.2 Closing the input stream

For web developers (non-normative)

**document.close()**

Closes the input stream that was opened by the document.open() method.

Throws an "InvalidStateError" DOMException if the Document is an XML document.

Throws an "InvalidStateError" DOMException if the parser is currently executing a custom element constructor.

The close() method must run the following steps:

1. If the Document object is an XML document, then throw an "InvalidStateError" DOMException.
2. If the Document object's throw-on-dynamic-markup-insertion counter is greater than zero, then throw an "InvalidStateError" DOMException.
3. If there is no script-created parser associated with the document, then return.
4. Insert an explicit "EOF" character at the end of the parser's input stream.
5. If there is a pending parsing-blocking script, then return.
6. 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.

### 8.4.3 document.write()  

For web developers (non-normative)

**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 input.
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. Users agents are explicitly allowed to avoid executing script elements inserted via this method. And to make matters even 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.

Throws an "InvalidStateError" DOMException when invoked on XML documents.

Throws an "InvalidStateError" DOMException if the parser is currently executing a custom element constructor.

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 steps, given a Document object document and a string input, are as follows:

1. If document is an XML document, then throw an "InvalidStateError" DOMException.
2. If document's throw-on-dynamic-markup-insertion counter is greater than 0, then throw an "InvalidStateError" DOMException.
3. If document's active parser was aborted is true, then return.
4. If the insertion point is undefined, then:
   1. If document's ignore-opens-during-unload counter is greater than 0 or document's ignore-destructive-writes counter is greater than 0, then return.
   2. Run the document open steps with document.
5. Insert input into the input stream just before the insertion point.
6. If there is no pending parsing-blocking script, have the HTML parser process input, one code point 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. If the parser pause flag is set, the tokenizer will abort immediately and no HTML will be parsed, per the tokenizer's parser pause flag check.

The document.write(...) method, when invoked, must run the document write steps with this Document object and a string that is the concatenation of all arguments passed.

8.4.4 document.writeln() throws an "InvalidStateError" DOMException when invoked on XML documents.

Throws an "InvalidStateError" DOMException if the parser is currently executing a custom element constructor.

For web developers (non-normative)

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.

Throws an "InvalidStateError" DOMException when invoked on XML documents.

Throws an "InvalidStateError" DOMException if the parser is currently executing a custom element constructor.
The DOMParser interface allows authors to create new Document objects by parsing strings, as either HTML or XML.

For web developers (non-normative)

```javascript
parser = new DOMParser()

// Constructs a new DOMParser object.

document = parser.parseFromString(string, type)

// Parses string using either the HTML or XML parser, according to type, and returns the resulting Document. type can be "text/html" (which will invoke the HTML parser), or any of "text/xml", "application/xml", "application/xhtml+xml", or "image/svg+xml" (which will invoke the XML parser).
```

Note

The design of DOMParser, as a class that needs to be constructed and then have its parseFromString() method called, is an unfortunate historical artifact. If we were designing this functionality today it would be a standalone function.

### IDL

```
[Exposed=Window]
interface DOMParser {
  constructor();

  [NewObject] Document parseFromString(DOMString string, DOMParserSupportedType type);
};
```

The new DOMParser() constructor steps are to do nothing.

The parseFromString(string, type) method steps are:

1. Let `document` be a new Document, whose content type is type and `url` is this's relevant global object's associated Document's URL.

Note

The document's encoding will be left as its default, of UTF-8. In particular, any XML declarations or `meta` elements found while parsing string will have no effect.

2. Switch on type:

   ➔ "text/html"

   1. Set document's type to "html".
   2. Create an HTML parser parser, associated with document.
   3. Place string into the input stream for parser. The encoding confidence is irrelevant.
   4. Start parser and let it run until it has consumed all the characters just inserted into the input stream.

Note

This might mutate the document's mode.
1. Create an XML parser[^1179] parse, associated with document, and with XML scripting support disabled[^1180].
2. Parse string using parser.
3. If the previous step resulted in an XML well-formedness or XML namespace well-formedness error, then:
   1. Assert: document has no child nodes.
   2. Let root be the result of creating an element given document, "parsererror", and "http://www.mozilla.org/newlayout/xml/parsererror.xml".
   3. Optionally, add attributes or children to root to describe the nature of the parsing error.
   4. Append root to document.

8.6 Timers[^959]

The setTimeout()[^959] and setInterval()[^959] methods allow authors to schedule timer-based callbacks.

```javascript
handle = self.setTimeout(handler [, timeout [, arguments... ] ])
Schedules a timeout to run handler after timeout milliseconds. Any arguments are passed straight through to the handler.

handle = self.setTimeout(code [, timeout ])
Schedules a timeout to compile and run code after timeout milliseconds.

self.clearTimeout(handle)
Cancels the timeout set with setTimeout()[^959] or setInterval()[^959] identified by handle.

handle = self.setInterval(handler [, timeout [, arguments... ] ])
Schedules a timeout to run handler every timeout milliseconds. Any arguments are passed straight through to the handler.

handle = self.setInterval(code [, timeout ])
Schedules a timeout to compile and run code every timeout milliseconds.

self.clearInterval(handle)
Cancels the timeout set with setInterval()[^959] or setTimeout()[^959] 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 WindowOrWorkerGlobalScope[^952] mixin have a list of active timers. Each entry in this list is identified by a number, which must be unique within the list for the lifetime of the object that implements the WindowOrWorkerGlobalScope[^952] mixin.

The setTimeout() method must return the value returned by the timer initialization steps[^969], passing them the method's arguments, the object on which the method for which the algorithm is running is implemented (a Window[^967] or WorkerGlobalScope[^1027] 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[^966], 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 `WindowOrWorkerGlobalScope` 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 `WindowOrWorkerGlobalScope` object on which the method was invoked, the method does nothing.)

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 an implementation-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 `initiating script` be the active script.
6. Assert: `initiating script` is not null, since this algorithm is always called from some script.
7. 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 these steps.
   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. If this throws an exception, catch it, and report the exception.
      → Otherwise
         1. Perform `HostEnsureCanCompileStrings`(`callerRealm, calleeRealm`). If this throws an exception, catch it, report the exception, and abort these steps.
         2. Let `script source` be the first method argument.
         3. Let `settings object` be method context's environment settings object.
         4. Let `base URL` be `initiating script`'s base URL.
         5. Let `fetch options` be a `script fetch options` whose `cryptographic nonce` is initiating script's `fetch options`'s `cryptographic nonce`, `integrity metadata` is the empty string, `parser metadata` is "not-parser-inserted", `credentials mode` is initiating script's `fetch options`'s `credentials mode`, and `referrer policy` is initiating script's `fetch options`'s `referrer policy`.
         6. Let `script` be the result of creating a classic script given `script source`, `settings object`, `base URL`, and `fetch options`.

Because `clearTimeout()` and `clearInterval()` clear entries from the same list, either method can be used to clear timers created by `setTimeout()` or `setInterval()`.
7. Run the classic script.

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.

8. Let timeout be the second method argument.

9. 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.

   \begin{itemize}
   \item \textbf{Note}\textit{ The task's timer nesting level is used both for nested calls to \texttt{setTimeout()}, and for the repeating timers created by \texttt{setInterval().} (Or, indeed, for any combination of the two.) In other words, it represents nested invocations of this algorithm, not of a particular method.}
   \end{itemize}

10. If timeout is less than 0, then set timeout to 0.

11. If nesting level is greater than 5, and timeout is less than 4, then set timeout to 4.

12. Increment nesting level by one.

13. Let task's timer nesting level be nesting level.

14. Return handle, and then continue running this algorithm in parallel.

15. 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).

16. 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.

   \begin{itemize}
   \item \textbf{Note}\textit{ Argument conversion as defined by Web IDL (for example, invoking to\texttt{String()} methods on objects passed as the first argument) happens in the algorithms defined in Web IDL, before this algorithm is invoked.}
   \end{itemize}

17. Optionally, wait a further implementation-defined length of time.

   \begin{itemize}
   \item \textbf{Note}\textit{ 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.}
   \end{itemize}

18. Queue a global task on the timer task source given method context to run task.

   \begin{itemize}
   \item \textbf{Note}\textit{ 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).}
   \end{itemize}
The `queueMicrotask(callback)` method must queue a microtask to invoke `callback`, and if `callback` throws an exception, report the exception.

The `queueMicrotask(callback)` method allows authors to schedule a callback on the microtask queue. This allows their code to run once the JavaScript execution context stack is next empty, which happens once all currently executing synchronous JavaScript has run to completion. This doesn't yield control back to the event loop, as would be the case when using, for example, `setTimeout(f, 0)`.

Authors ought to be aware that scheduling a lot of microtasks has the same performance downsides as running a lot of synchronous code. Both will prevent the browser from doing its own work, such as rendering. In many cases, `requestAnimationFrame()` or `requestIdleCallback()` is a better choice. In particular, if the goal is to run code before the next rendering cycle, that is the purpose of `requestAnimationFrame()`.

As can be seen from the following examples, the best way of thinking about `queueMicrotask(callback)` is as a mechanism for rearranging synchronous code, effectively placing the queued code immediately after the currently executing synchronous JavaScript has run to completion.

**Example**

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:

```javascript
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
```

**8.7 Microtask queuing**

For web developers (non-normative)

The `queueMicrotask(callback)` method must queue a microtask to invoke `callback`, and if `callback` throws an exception, report the exception.

The `queueMicrotask(callback)` method allows authors to schedule a callback on the microtask queue. This allows their code to run once the JavaScript execution context stack is next empty, which happens once all currently executing synchronous JavaScript has run to completion. This doesn't yield control back to the event loop, as would be the case when using, for example, `setTimeout(f, 0)`.

Authors ought to be aware that scheduling a lot of microtasks has the same performance downsides as running a lot of synchronous code. Both will prevent the browser from doing its own work, such as rendering. In many cases, `requestAnimationFrame()` or `requestIdleCallback()` is a better choice. In particular, if the goal is to run code before the next rendering cycle, that is the purpose of `requestAnimationFrame()`.

As can be seen from the following examples, the best way of thinking about `queueMicrotask(callback)` is as a mechanism for rearranging synchronous code, effectively placing the queued code immediately after the currently executing synchronous JavaScript has run to completion.

**Example**

The most common reason for using `queueMicrotask(callback)` is to create consistent ordering, even in the cases where information is available synchronously, without introducing undue delay.

For example, consider a custom element firing a load event, that also maintains an internal cache of previously-loaded data. A naïve implementation might look like:

```javascript
MyElement.prototype.loadData = function (url) {
  if (this._cache[url]) {
```
This naïve implementation is problematic, however, in that it causes its users to experience inconsistent behavior. For example, code such as

```javascript
element.addEventListener("load", () => console.log("loaded"));
console.log("1");
element.loadData();
console.log("2");
```

will sometimes log "1, 2, loaded" (if the data needs to be fetched), and sometimes log "1, loaded, 2" (if the data is already cached). Similarly, after the call to `loadData()`, it will be inconsistent whether or not the data is set on the element.

To get a consistent ordering, `queueMicrotask()` can be used:

```javascript
MyElement.prototype.loadData = function (url) {
    if (this._cache[url]) {
        queueMicrotask(() => {
            this._setData(this._cache[url]);
            this.dispatchEvent(new Event("load"));
        });
    } else {
        fetch(url).then(res => res.arrayBuffer()).then(data => {
            this._cache[url] = data;
            this._setData(data);
            this.dispatchEvent(new Event("load"));
        });
    }
};
```

By essentially rearranging the queued code to be after the JavaScript execution context stack empties, this ensures a consistent ordering and update of the element’s state.

---

**Example**

Another interesting use of `queueMicrotask()` is to allow uncoordinated “batching” of work by multiple callers. For example, consider a library function that wants to send data somewhere as soon as possible, but doesn’t want to make multiple network requests if doing so is easily avoidable. One way to balance this would be like so:

```javascript
const queuedToSend = [];

function sendData(data) {
    queuedToSend.push(data);

    if (queuedToSend.length === 1) {
        queueMicrotask(() => {
            const stringToSend = JSON.stringify(queuedToSend);
            queuedToSend.length = 0;

            fetch("/endpoint", stringToSend);
        });
    }
};
```
With this architecture, multiple subsequent calls to `sendData()` within the currently executing synchronous JavaScript will be batched together into one `fetch()` call, but with no intervening event loop tasks preempting the fetch (as would have happened with similar code that instead used `setTimeout()`).

### 8.8 User prompts

#### 8.8.1 Simple dialogs

For web developers (non-normative)

`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 control 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.

The `alert()` and `alert(message)` method steps are:

1. If we cannot show simple dialogs for this, then return.
2. If the method was invoked with no arguments, then let `message` be the empty string; otherwise, let `message` be the method’s first argument.
3. Set `message` to the result of normalizing newlines given `message`.
4. Set `message` to the result of optionally truncating message.
5. Show `message` to the user, treating U+000A LF as a line break.
6. Optionally, pause while waiting for the user to acknowledge the message.

**Note**

This method is defined using two overloads, instead of using an optional argument, for historical reasons. The practical impact of this is that alert(undefined) is treated as alert("undefined"), but alert() is treated as alert("").

The `confirm(message)` method steps are:

1. If we cannot show simple dialogs for this, then return false.
2. Set `message` to the result of normalizing newlines given `message`.
3. Set `message` to the result of optionally truncating message.
4. Show `message` to the user, treating U+000A LF as a line break, and ask the user to respond with a positive or negative response.
5. Pause until the user responds either positively or negatively.

6. If the user responded positively, return true; otherwise, the user responded negatively: return false.

The prompt(message, default) method steps are:

1. If we cannot show simple dialogs for this, then return null.

2. Set message to the result of normalizing newlines given message.

3. Set message to the result of optionally truncating message.

4. Set default to the result of optionally truncating default.

5. Show message to the user, treating U+000A LF as a line break, and ask the user to either respond with a string value or abort. The response must be defaulted to the value given by default.

6. Pause while waiting for the user's response.

7. If the user aborts, then return null; otherwise, return the string that the user responded with.

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!".

We cannot show simple dialogs for a Window window when the following algorithm returns true:

1. If the active sandboxing flag set of window's associated Document has the sandboxed modals flag set, then return true.

2. If window's relevant settings object's origin and window's relevant settings object's top-level origin are not same origin-domain, then return true.

3. If window's relevant agent's event loop's termination nesting level is nonzero, then optionally return true.

4. Optionally, return true. (For example, the user agent might give the user the option to ignore all modal dialogs, and would thus abort at this step whenever the method was invoked.)

5. Return false.

8.8.2 Printing

For web developers (non-normative)

window . print() Prompts the user to print the page.

The print() method steps are:

1. Let document be this's associated Document.

2. If document is ready for post-load tasks, then run the printing steps for document.

3. Otherwise, set document's print when loaded flag.

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 for a Document document are:
1. The user agent may display a message to the user or return (or both).

Example

For instance, a kiosk browser could silently ignore any invocations of the `print()` method.

Example

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 document has the sandboxed modals flag set, then return.

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 an event named beforeprint at the relevant global object of document, as well as any child browsing contexts in it.

Example

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 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 an event named afterprint at the relevant global object of document, as well as any child browsing contexts in it.

Example

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.

8.9 System state and capabilities

8.9.1 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 handlers:

```
IDL [Exposed=Window]
interface Navigator {
    // objects implementing this interface also implement the interfaces given below
};
```

These interface mixins are defined separately so that WorkerNavigator can re-use parts of the Navigator interface.
8.9.1.1 Client identification

```
interface mixin NavigatorID {
readonly attribute DOMString appCodeName; // constant "Mozilla"
readonly attribute DOMString appName; // constant "Netscape"
readonly attribute DOMString appVersion;
readonly attribute DOMString platform;
readonly attribute DOMString product; // constant "Gecko"
[Exposed=Window] readonly attribute DOMString productSub;
readonly attribute DOMString userAgent;
[Exposed=Window] readonly attribute DOMString vendor;
[Exposed=Window] readonly attribute DOMString vendorSub; // constant ""
};
```

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.

The user agent has a **navigator compatibility mode**, which is either Chrome, Gecko, or WebKit.

**Note**
The **navigator compatibility mode** constrains the **NavigatorID** interface to the combinations of attribute values and presence of `taintEnabled()` and `oscpu` that are known to be compatible with existing web content.

Client detection should always be limited to detecting known current versions; future versions and unknown versions should always be assumed to be fully compliant.

**For web developers (non-normative)**

- `self . navigator.appCodeName` Returns the string "Mozilla".
- `self . navigator.appName` Returns the string "Netscape".
- `self . navigator.appVersion` Returns the version of the browser.
- `self . navigator.platform` Returns the name of the platform.
- `self . navigator.product` Returns the string "Gecko".
- `window . navigator.productSub` Returns either the string "20030107", or the string "20100101".
- `self . navigator.userAgent` Returns the complete `User-Agent` header.
- `window . navigator.vendor` Returns either the empty string, the string "Apple Computer, Inc.", or the string "Google Inc.".
- `window . navigator.vendorSub` Returns the empty string.

**appCodeName**
Must return the string "Mozilla".

**appName**
Must return the string "Netscape".
appVersion
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
Must return either the empty string or a string representing the platform on which the browser is executing, e.g. "MacIntel", "Win32", "FreeBSD 1386", "WebTV OS".

product
Must return the string "Gecko".

productSub
Must return the appropriate string from the following list:

- If the navigator compatibility mode\[^{p967}\] is Chrome or WebKit
  The string "20030107".

- If the navigator compatibility mode\[^{p967}\] is Gecko
  The string "20100101".

userAgent
Must return the default `User-Agent` value.

vendor
Must return the appropriate string from the following list:

- If the navigator compatibility mode\[^{p967}\] is chrome
  The string "Google Inc.".

- If the navigator compatibility mode\[^{p967}\] is Gecko
  The empty string.

- If the navigator compatibility mode\[^{p967}\] is WebKit
  The string "Apple Computer, Inc.".

vendorSub
Must return the empty string.

If the navigator compatibility mode\[^{p967}\] is Gecko, then the user agent must also support the following partial interface:

```idl
partial interface mixin NavigatorID {
  [Exposed=Window] boolean taintEnabled(); // constant false
  [Exposed=Window] readonly attribute DOMString oscpu;
};
```

The `taintEnabled()` method must return false.

The `oscpu` attribute's getter must return either the empty string or a string representing the platform on which the browser is executing, e.g. "Windows NT 10.0; Win64; x64", "Linux x86_64".

⚠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 implementers are strongly urged to include as little information in this API as possible.

8.9.1.2 Language preferences\[^{p96}\]

```idl
interface mixin NavigatorLanguage {
  readonly attribute DOMString language;
  readonly attribute FrozenArray<DOMString> languages;
};
```
For web developers (non-normative)

```javascript
self.navigator.language
```
Returns a language tag representing the user's preferred language.

```javascript
self.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**

Must return a valid BCP 47 language tag representing either a plausible language or the user's most preferred language. ([BCP47](https://www.rfc-editor.org/rfc/rfc4646))

**Languages**

Must return a `frozen 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](https://www.rfc-editor.org/rfc/rfc4646))

Whenever the user agent needs to make the `navigator.languages` attribute of a `Window` or `WorkerGlobalScope` object global return a new set of language tags, the user agent must queue a global task on the DOM manipulation task source given global to fire an event named `languagechange` at `global`, and wait until that task begins to be executed before actually returning a new value.

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](https://www.torproject.org/))
- 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.

### 8.9.1.3 Browser state

```idl```
```
interface mixin NavigatorOnLine {
  readonly attribute boolean online;
};
```

For web developers (non-normative)

```javascript
self.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 `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` global changes from true to false, the user agent must queue a global task on the networking task source given global to fire an event named `offline` at global.

On the other hand, when the value that would be returned by the `navigator.onLine` attribute of a `Window` or `WorkerGlobalScope` global changes from false to true, the user agent must queue a global task on the networking task source given global to fire an event named `online` at the `Window` or `WorkerGlobalScope` object.

---

**Example**

In this example, an indicator is updated as the browser goes online and offline.

```html
<!DOCTYPE HTML>
<html lang="en">
<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 id="indicator">(state unknown)</span></p>
</body>
</html>
```

---

### 8.9.1.4 Custom scheme handlers: the `registerProtocolHandler()` method

**IDL**

```idl
interface mixin NavigatorContentUtils {
    [SecureContext] undefined registerProtocolHandler(DOMString scheme, USVString url);
    [SecureContext] undefined unregisterProtocolHandler(DOMString scheme, USVString url);
};
```

---

**For web developers (non-normative)**

**`window.navigator.registerProtocolHandler()`**

Registers a handler for `scheme` at `url`. 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, they are given the opportunity to use that web site. [SMS]

The string "%s" in `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 `url`.

**`window.navigator.unregisterProtocolHandler()`**

Unregisters the handler given by the arguments.

- Throws a `SecurityError` `DOMException` if the user agent blocks the deregistration (this might happen if with invalid schemes, for instance).
- Throws a `SyntaxError` `DOMException` if the "%s" string is missing in `url`.

The `registerProtocolHandler(scheme, url)` method steps are:

1. Let (normalizedScheme, normalizedURLString) be the result of running `normalize protocol handler parameters` with `scheme`, `url`, and this's relevant settings object.
2. In parallel: register a handler for `normalizedScheme` and `normalizedURLString`. User agents may, within the constraints described, do whatever they like. 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 handlers their default, or cancel the request. User agents could also 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.

When the user agent uses this handler for a URL `inputURL`:

1. Assert: `inputURL`'s `scheme` is `normalizedScheme`.
2. Let `inputURLString` be the serialization of `inputURL`.
3. Let `encodedURL` be the result of running `UTF-8 percent-encode` on `inputURLString` using the `component percent-encode set`.
4. Let `handlerURLString` be `normalizedURLString`.
5. Replace the first instance of "%s" in `handlerURLString` with `encodedURL`.
6. Let `resultURL` be the result of parsing `handlerURLString`.
7. Navigate an appropriate browsing context to `resultURL`.

Example

If the user had visited a site at `https://example.com/` that made the following call:

```javascript
navigator.registerProtocolHandler('web+soup', 'soup?url=%s')
```

...and then, much later, while visiting `https://www.example.net/`, clicked on a link such as:

```html
<a href="web+soup:chicken-kïwi">Download our Chicken Kïwi soup!</a>
```

...then the UA might navigate to the following URL:

`https://example.com/soup?url=web+soup:chicken-k%C3%AFwi`

This site could then do whatever it is that it does with soup (synthesize it and ship it to the user, or whatever).

This does not define when the handler is used. To some extent, the processing model for navigating across documents defines some cases where it is relevant, but in general user agents may use this information wherever they would otherwise consider handing schemes to native plugins or helper applications.

The `unregisterProtocolHandler(scheme, url)` method steps are:

1. Let `(normalizedScheme, normalizedURLString)` be the result of running `normalize protocol handler parameters` with `scheme, url, and this's relevant settings object`.
2. In parallel: unregister the handler described by `normalizedScheme` and `normalizedURLString`.

To normalize protocol handler parameters, given a string `scheme`, a string `url`, and an environment settings object, run these steps:

1. Set `scheme` to `scheme`, converted to ASCII lowercase.
2. If `scheme` is neither a safelisted scheme nor a string starting with "web+" followed by one or more ASCII lower alphas, then throw a "SecurityError" DOMException.

Note

This means that including a colon in scheme (as in "mailto:" will throw.

The following schemes are the safelisted schemes:

- bitcoin
Custom scheme handlers 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 an HTTP(S) scheme, 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 handler for web+spam: -- 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.

**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 private data.** Web page authors may reference a custom scheme handler using URL data considered private. They might do so with the expectation that the user's choice of handler points to a page inside the organization, ensuring that sensitive data will not be exposed to third parties. However, a user may have registered a handler pointing to an external site, resulting in a data leak to that third party. Implementors might wish to consider allowing administrators to disable custom handlers on certain subdomains, content types, 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.

8.9.1.5 Cookies

IDL interface mixin NavigatorCookies {
  readonly attribute boolean cookieEnabled;
};

For web developers (non-normative)

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 HTTP State Management Mechanism, and false if it ignores cookie change requests. [COOKIES]

8.10 Images

IDL [Exposed=(Window, Worker), Serializable, Transferable]

interface ImageBitmap {
  readonly attribute unsigned long width;
  readonly attribute unsigned long height;
  undefined close();
};

typedef (CanvasImageSource or Blob or ImageData) ImageBitmapSource;

enum ImageOrientation { "none", "flipY" };
enum PremultiplyAlpha { "none", "premultiply", "default" };
enum ColorSpaceConversion { "none", "default" };
enum ResizeQuality { "pixelated", "low", "medium", "high" };

dictionary ImageBitmapOptions {
  ImageOrientation imageOrientation = "none";
  PremultiplyAlpha premultiplyAlpha = "default";
  ColorSpaceConversion colorSpaceConversion = "default";
  [EnforceRange] unsigned long resizeWidth;
  [EnforceRange] unsigned long resizeHeight;
  ResizeQuality resizeQuality = "low";
};

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
An `ImageBitmap` object whose [[Detached]] internal slot value is false 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()`.

`ImageBitmap` objects are **serializable objects** and **transferable objects**.

Their serialization steps, given `value` and `serialized`, are:

1. If `value`'s `origin-clean` flag is not set, then throw a "DataCloneError" DOMException.
2. Set `serialized.[[BitmapData]]` to a copy of `value`'s `bitmap data`.

Their deserialization steps, given `serialized` and `value`, are:

1. Set `value`.`bitmap data` to `serialized.[[BitmapData]]`.

Their transfer steps, given `value` and `dataHolder`, are:

1. If `value`'s `origin-clean` flag is not set, then throw a "DataCloneError" DOMException.
2. Set `dataHolder.[[BitmapData]]` to `value`'s `bitmap data`.
3. Unset `value`.`bitmap data`.

Their transfer-receiving steps, given `dataHolder` and `value`, are:

```javascript
for (let x = 0; x < width; x++) {
  for (let y = 0; y < height; y++) {
    let data = value.data.getPixel(x, y);
    dataHolder.data.setPixel(x, y, data);
  }
}
```

For web developers (non-normative)
1. Set value's bitmap data to `dataHolder.[[BitmapData]]`.

The `createImageBitmap(image, options)` and `createImageBitmap(image sx, sy, sw, sh, options)` methods, when invoked, must run these steps:

1. Let `p` be a new promise.
2. If either `sw` or `sh` is given and is 0, then return `p` rejected with a `RangeError`.
3. If either `options`'s `resizeWidth` or `options`'s `resizeHeight` is present and is 0, then return `p` rejected with an "InvalidStateError" `DOMException`.
4. Check the usability of the `image` argument. If this throws an exception or returns `bad`, then return `p` rejected with an "InvalidStateError" `DOMException`.
5. Let `imageBitmap` be a new `ImageBitmap` object.
6. Switch on `image`:
   - `img`
     1. If `image`'s media data has no intrinsic dimensions (e.g., it's a vector graphic with no specified content size) and either `options`'s `resizeWidth` or `options`'s `resizeHeight` is not present, then return `p` rejected with an "InvalidStateError" `DOMException`.
     2. If `image`'s media data has no intrinsic dimensions (e.g., it's a vector graphics with no specified content size), it should be rendered to a bitmap of the size specified by the `resizeWidth` and the `resizeHeight` options.
     3. Set `imageBitmap`'s bitmap data to a copy of `image`'s media data, cropped to the source rectangle with formatting. If this is an animated image, `imageBitmap`'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.
     4. If the `origin` of `image`'s image is not same origin with `entry settings object`'s `origin`, then set the `origin-clean` flag of `imageBitmap`'s bitmap to false.
   - `video`
     1. If `image`'s `networkState` attribute is `NETWORK_EMPTY`, then return `p` rejected with an "InvalidStateError" `DOMException`.
     2. Set `imageBitmap`'s bitmap data to 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 with formatting.
     3. If the `origin` of `image`'s video is not same origin with `entry settings object`'s `origin`, then set the `origin-clean` flag of `imageBitmap`'s bitmap to false.
   - `canvas`
     1. Set `imageBitmap`'s bitmap data to a copy of `image`'s bitmap data, cropped to the source rectangle with formatting.
     2. Set the `origin-clean` flag of the `imageBitmap`'s bitmap to the same value as the `origin-clean` flag of `image`'s bitmap.
     3. Run this step in parallel:

975
1. Resolve \( p \) with `imageBitmap`.

\[ \text{Blob} \]

Run these step \textit{in parallel}\(^{42} \):

1. Let `imageData` be the result of reading `image`'s data. If an \textit{error} occurs during reading of the object\(^{47} \), then reject \( p \) with an "\textit{InvalidStateError}" \texttt{DOMException} and abort these steps.

2. Apply the \textit{image sniffing rules} to determine the file format of `imageData`, with MIME type of `image` (as given by `image`'s \textit{type} attribute) giving the official type.

3. If `imageData` is not in a supported image file format (e.g., it's not an image at all), or if `imageData` is corrupted in some fatal way such that the image dimensions cannot be obtained (e.g., a vector graphic with no intrinsic size), then reject \( p \) with an "\textit{InvalidStateError}" \texttt{DOMException} and abort these steps.

4. Set `imageBitmap`'s \textit{bitmap data}\(^{974} \) to `imageData`, \textit{cropped to the source rectangle with formatting}\(^{976} \). If this is an animated image, `imageBitmap`'s \textit{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.

5. Resolve \( p \) with `imageBitmap`.

\[ \text{ImageData}\(^{942} \]

1. Let `buffer` be `image`'s \textit{data} attribute value's \([\text{ViewedArrayBuffer}]\) internal slot.

2. If \texttt{IsDetachedBuffer}(`buffer`) is true, then return \( p \) rejected with an "\textit{InvalidStateError}" \texttt{DOMException}.

3. Set `imageBitmap`'s \textit{bitmap data} to `image`'s image data, \textit{cropped to the source rectangle with formatting}\(^{976} \).

4. Run this step \textit{in parallel}\(^{42} \):

1. Resolve \( p \) with `imageBitmap`.

\[ \text{ImageBitmap}\(^{973} \]

1. Set `imageBitmap`'s \textit{bitmap data} to a copy of `image`'s \textit{bitmap data}, \textit{cropped to the source rectangle with formatting}\(^{976} \).

2. Set the \textit{origin-clean} flag of `imageBitmap`'s bitmap to the same value as the \textit{origin-clean} flag of `image`'s bitmap.

3. Run this step \textit{in parallel}\(^{42} \):

1. Resolve \( p \) with `imageBitmap`.

7. Return \( p \).

When the steps above require that the user agent \textit{crop bitmap data to the source rectangle with formatting}, the user agent must run the following steps:

1. Let \textit{input} be the \textit{bitmap data} being transformed.

2. If \( sx, sy, sw \) and \( sh \) are specified, let \textit{sourceRectangle} be a rectangle whose corners are the four points \((sx, sy), (sx+sw, sy), (sx+sw, sy+sh), (sx, sy+sh)\). Otherwise let \textit{sourceRectangle} be a rectangle whose corners are the four points \((0,0), (\text{width of input}, 0), (\text{width of input}, \text{height of input}), (0, \text{height of input})\).

\[ \text{Note} \]

\textbf{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.} \[ 976 \]

3. Let \textit{outputWidth} be determined as follows:

\[ \text{If the \textbf{resizeWidth} member of options is specified} \]

the value of the \textbf{resizeWidth} member of options

\[ \text{If the \textbf{resizeWidth} member of options is not specified, but the \textbf{resizeHeight} member is specified} \]

the width of \textit{sourceRectangle}, times the value of the \textbf{resizeHeight} member of options, divided by the height of \textit{sourceRectangle}, rounded up to the nearest integer
4. Let `outputHeight` be determined as follows:
   - If the `resizeHeight` member of `options` is specified
     the value of the `resizeHeight` member of `options`
   - If the `resizeHeight` member of `options` is not specified, but the `resizeWidth` member is specified
     the height of `sourceRectangle`, times the value of the `resizeWidth` member of `options`, divided by the width of `sourceRectangle`, rounded up to the nearest integer
   - If neither `resizeWidth` nor `resizeHeight` are specified
     the height of `sourceRectangle`

5. 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.

6. Let `output` be the rectangle on the plane denoted by `sourceRectangle`.

7. Scale `output` to the size specified by `outputWidth` and `outputHeight`. The user agent should use the value of the `resizeQuality` option to guide the choice of scaling algorithm.

8. If the value of the `imageOrientation` member of `options` is "flipY", `output` must be flipped vertically, disregarding any image orientation metadata of the source (such as EXIF metadata), if any. [EXIF]

   **Note**
   If the value is "none", no extra step is required.

9. If `image` is an `img` element or a `Blob` object, let `val` be the value of the `colorSpaceConversion` member of `options`, and then run these substeps:
   1. If `val` is "default", the color space conversion behavior is implementation-specific, and should be chosen according to the color space that the implementation uses for drawing images onto the canvas.
   2. If `val` is "none", `output` must be decoded without performing any color space conversions. This means that the image decoding algorithm must ignore color profile metadata embedded in the source data as well as the display device color profile.

   **Note**
   The native color space of canvas is currently unspecified, but this is expected to change in the future.

10. Let `val` be the value of `premultiplyAlpha` member of `options`, and then run these substeps:
   1. If `val` is "default", the alpha premultiplication behavior is implementation-specific, and should be chosen according to implementation deems optimal for drawing images onto the canvas.
   2. If `val` is "premultiply", the output that is not premultiplied by alpha must have its color components multiplied by alpha and that is premultiplied by alpha must be left untouched.
   3. If `val` is "none", the output that is not premultiplied by alpha must be left untouched and that is premultiplied by alpha must have its color components divided by alpha.

11. Return `output`.

When the `close()` method is called, the user agent must run these steps:
   1. Set this `ImageBitmap` object's `[[Detached]]` internal slot value to true.
   2. Unset this `ImageBitmap` object's `bitmap data`.

The `width` attribute's getter must run these steps:
   1. If this `ImageBitmap` object's `[[Detached]]` internal slot's value is true, then return 0.
   2. Return this `ImageBitmap` object's width, in CSS pixels.
The `height` attribute's getter must run these steps:

1. If this `ImageBitmap` object's `[[Detached]]` internal slot's value is true, then return 0.
2. Return this `ImageBitmap` object's height, in CSS pixels.

The `ResizeQuality` enumeration is used to express a preference for the interpolation quality to use when scaling images.

The "pixelated" value indicates a preference to scale the image that maximizes the appearance. Scaling algorithms that "smooth" colors are acceptable, such as bilinear interpolation.

The "low" value indicates a preference for a low level of image interpolation quality. Low-quality image interpolation may be more computationally efficient than higher settings.

The "medium" value indicates a preference for a medium level of image interpolation quality.

The "high" value indicates a preference for a high level of image interpolation quality. High-quality image interpolation may be more computationally expensive than lower settings.

**Note**

Bilinear scaling is an example of a relatively fast, lower-quality image-smoothing algorithm. Bicubic or Lanczos scaling are examples of image-scaling algorithms that produce higher-quality output. This specification does not mandate that specific interpolation algorithms be used unless the value is "pixelated".

**Example**

Using this API, a sprite sheet can be precut and prepared:

```javascript
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.person = image }),
         createImageBitmap(image, 40, 0, 40, 40).then(function (image) { sprites.grass = 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);
```

8.11 Animation frames

Some objects include the `AnimationFrameProvider` interface mixin.
Each `AnimationFrameProvider` object also has a `target object` that stores the provider's internal state. It is defined as follows:

**If the `AnimationFrameProvider` is a `Window`**

The `Window`'s associated `Document` object.

**If the `AnimationFrameProvider` is a `DedicatedWorkerGlobalScope`**

The `DedicatedWorkerGlobalScope` object.

Each `target object` has a `map of animation frame callbacks`, which is an `ordered map` that must be initially empty, and an `animation frame callback identifier`, which is a number that must initially be zero.

An `AnimationFrameProvider` provider is considered `supported` if any of the following hold:

- `provider` is a `Window`.
- `provider`'s `owner set` contains a `Document` object.
- Any of the `DedicatedWorkerGlobalScope` objects in `provider`'s `owner set` are `supported`.

The `requestAnimationFrame(callback)` method must run the following steps:

1. If this `AnimationFrameProvider` is not supported, then throw a "`NotSupportedError`" `DOMException`.
2. Let `target` be this `AnimationFrameProvider`'s `target object`.
3. Increment `target`'s `animation frame callback identifier` by one, and let `handle` be the result.
4. Let `callbacks` be `target`'s `map of animation frame callbacks`.
5. Set `callbacks[handle]` to `callback`.
6. Return `handle`.

The `cancelAnimationFrame(handle)` method must run the following steps:

1. If this `AnimationFrameProvider` is not supported, then throw a "`NotSupportedError`" `DOMException`.
2. Let `callbacks` be this `AnimationFrameProvider`'s `target object`'s `map of animation frame callbacks`.

To run the animation frame callbacks for a `target object` target with a timestamp `now`:

1. Let `callbacks` be target's `map of animation frame callbacks`.
2. Let `callbackHandles` be the result of `getting the keys` of `callbacks`.
3. For each `handle` in `callbackHandles`, if `handle` exists in `callbacks`:
   1. Let `callback` be `callbacks[handle]`.
   2. Remove `callbacks[handle]`.
   3. Invoke `callback`, passing `now` as the only argument, and if an exception is thrown, report the exception.

Example

Inside workers, `requestAnimationFrame()` can be used together with an `OffscreenCanvas` transferred from a canvas element. First, in the document, transfer control to the worker:
```javascript
const offscreenCanvas = document.getElementById("c").transferControlToOffscreen();
worker.postMessage(offscreenCanvas, [offscreenCanvas]);

Then, in the worker, the following code will draw a rectangle moving from left to right:

```javascript
let ctx, pos = 0;
function draw(dt) {
  ctx.clearRect(0, 0, 100, 100);
  ctx.fillRect(pos, 0, 10, 10);
  pos += 10 * dt;
  requestAnimationFrame(draw);
}

self.onmessage = function(ev) {
  const transferredCanvas = ev.data;
  ctx = transferredCanvas.getContext("2d");
  draw();
};
```
9 Communication §p98

9.1 The MessageEvent interface §p98

Messages in server-sent events, web sockets, cross-document messaging, channel messaging, and broadcast channels, use the MessageEvent interface for their message events:

```idl
[Exposed=(Window,Worker,AudioWorklet)]
interface MessageEvent : Event {
    readonly attribute DOMString type, optional MessageEventInit eventInitDict = {};
    readonly attribute any data;
    readonly attribute USVString origin;
    readonly attribute DOMString lastEventId;
    readonly attribute MessageEventSource? source;
    readonly attribute FrozenArray<MessagePort> ports;

    undefined initMessageEvent(DOMString type, optional boolean bubbles = false, optional boolean cancelable = false, optional any data = null, optional USVString origin = "", optional DOMString lastEventId = "," optional MessageEventSource? source = null, optional sequence<MessagePort> ports = []);
};

dictionary MessageEventInit : EventInit {
    any data = null;
    USVString origin = "";
    DOMString lastEventId = "";
    MessageEventSource? source = null;
    sequence<MessagePort> ports = [];
};
typedef (WindowProxy or MessagePort or ServiceWorker) MessageEventSource;
```

For web developers (non-normative)

- **event.data** Returns the data of the message.
- **event.origin** Returns the origin of the message, for server-sent events and cross-document messaging.
- **event.lastEventId** Returns the last event ID string, for server-sent events.
- **event.source** Returns the WindowProxy of the source window, for cross-document messaging, and the MessagePort being attached, in the connect event fired at SharedWorkerGlobalScope objects.
- **event.ports** Returns the MessagePort array sent with the message, for cross-document messaging and channel messaging.

The `data` attribute must return the value it was initialized to. It represents the message being sent.

The `origin` attribute must return the value it was initialized to. It represents, in server-sent events and cross-document messaging, the `origin` of the document that sent the message (typically the scheme, hostname, and port of the document, but not its path or fragment).

The `lastEventId` attribute must return the value it was initialized to. It represents, in server-sent events, the `last event ID string` of the event source.
The `source` attribute must return the value it was initialized to. It represents, in cross-document messaging, the `WindowProxy` of the `Window` object from which the message came; and in the `connect` events used by `shared workers`, the newly connecting `MessagePort`.

The `ports` attribute must return the value it was initialized to. It represents, in cross-document messaging and channel messaging, the `MessagePort` array being sent.

The `initMessageEvent()` method must initialize the event in a manner analogous to the similarly-named `initEvent()` method.

[DOM]

This section is non-normative.

To enable servers to push data to web pages over HTTP or using dedicated server-push protocols, this specification introduces the `EventSource` interface.

Using this API consists of creating an `EventSource` object and registering an event listener.

```javascript
var source = new EventSource('updates.cgi');
source.onmessage = function (event) {
  alert(event.data);
};
```

On the server-side, the script ("updates.cgi" in this case) sends messages in the following form, with the `text/event-stream` MIME type:

```
data: This is the first message.
data: This is the second message, it has two lines.
data: This is the third message.
```

Authors can separate events by using different event types. Here is a stream that has two event types, "add" and "remove":

```
event: add
data: 73857293

event: remove
data: 2153

event: add
data: 113411
```

The script to handle such a stream would look like this (where `addHandler` and `removeHandler` are functions that take one argument, the event):

```javascript
var source = new EventSource('updates.cgi');
source.addEventListener('add', addHandler, false);
```
source.addEventListener('remove', removeHandler, false);

The default event type is "message".

Event streams are always decoded as UTF-8. There is no way to specify another character encoding.

Event stream requests can be redirected using HTTP 301 and 307 redirects as with normal HTTP requests. Clients will reconnect if the connection is closed; a client can be told to stop reconnecting using the HTTP 204 No Content response code.

Using this API rather than emulating it using XMLHttpRequest or an iframe allows the user agent to make better use of network resources in cases where the user agent implementer and the network operator are able to coordinate in advance. Amongst other benefits, this can result in significant savings in battery life on portable devices. This is discussed further in the section below on connectionless push.

9.2.2 The EventSource interface

```idl
[Exposed=({Window,Worker})]
interface EventSource : EventTarget {
  constructor(USVString url, optional EventSourceInit eventSourceInitDict = {});

  readonly attribute USVString url;
  readonly attribute boolean withCredentials;

  // ready state
  const unsigned short CONNECTING = 0;
  const unsigned short OPEN = 1;
  const unsigned short CLOSED = 2;
  readonly attribute unsigned short readyState;

  // networking
  attributeEventHandler onopen;
  attributeEventHandler onmessage;
  attribute EventHandler onerror;
  undefined close();
};
```

dictionary EventSourceInit {
  boolean withCredentials = false;
};

Each EventSource object has the following associated with it:

- A url (a URL record). Set during construction.
- A request. This must initially be null.
- A reconnection time, in milliseconds. This must initially be an implementation-defined value, probably in the region of a few seconds.
- A last event ID string. This must initially be the empty string.

Apart from url these are not currently exposed on the EventSource object.

For web developers (non-normative)

```javascript
source = new EventSource('url', { withCredentials: true })

Creates a new EventSource object.

url is a string giving the URL that will provide the event stream.

Setting withCredentials to true will set the credentials mode for connection requests to url to "include".
```
source . close
Aborts any instances of the fetch algorithm started for this EventSource object, and sets the readyState attribute to CLOSED.

source . url
Returns the URL providing the event stream.

source . withCredentials
Returns true if the credentials mode for connection requests to the URL providing the event stream is set to "include", and false otherwise.

source . readyState
Returns the state of this EventSource object's connection. It can have the values described below.

The EventSource(url, eventSourceInitDict) constructor, when invoked, must run these steps:

1. Let ev be a new EventSource object.
2. Let settings be ev's relevant settings object.
3. Let urlRecord be the result of parsing url with settings's API base URL and settings's API URL character encoding.
4. If urlRecord is failure, then throw a "SyntaxError" DOMException.
5. Set ev's url to urlRecord.
7. If the value of eventSourceInitDict's withCredentials member is true, then set corsAttributeState to Use Credentials and set ev's withCredentials attribute to true.
8. Let request be the result of creating a potential-CORS request given urlRecord, the empty string, and corsAttributeState.
9. Set request's client to settings.
10. User agents may set 'Accept' / text/event-stream in request's header list.
11. Set request's cache mode to "no-store".
12. Set ev's request to request.
13. Run this step in parallel:
   1. Fetch request.

The url attribute's getter must return the serialization of this EventSource object's url.

The withCredentials attribute must return the value to which it was last initialized. When the object is created, it must be initialized to false.

The readyState attribute represents the state of the connection. It can have the following values:

**CONNECTING (numeric value 0)**

The connection has not yet been established, or it was closed and the user agent is reconnecting.

**OPEN (numeric value 1)**

The user agent has an open connection and is dispatching events as it receives them.

**CLOSED (numeric value 2)**

The connection is not open, and the user agent is not trying to reconnect. Either there was a fatal error or the close() method was invoked.

When the object is created its readyState must be set to CONNECTING (0). The rules given below for handling the connection definition when the value changes.
The `close()` method must abort any instances of the `fetch` algorithm started for this `EventSource` object, and must set the `readyState` attribute to `CLOSED`.

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 `EventSource` interface:

<table>
<thead>
<tr>
<th>Event handler</th>
<th>Event handler event type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>onopen</code></td>
<td><code>open</code></td>
</tr>
<tr>
<td><code>onmessage</code></td>
<td><code>message</code></td>
</tr>
<tr>
<td><code>onerror</code></td>
<td><code>error</code></td>
</tr>
</tbody>
</table>

### 9.2.3 Processing model

The resource indicated in the argument to the `EventSource` constructor is fetched when the constructor is run.

As data is received, the `tasks` queued by the `networking task source` to handle the data must act as follows.

HTTP 200 OK responses with a `Content-Type` header specifying the type `text/event-stream`, ignoring any `MIME type` parameters, must be processed line by line as described below.

When a successful response with a supported `MIME type` is received, such that the user agent begins parsing the contents of the stream, the user agent must announce the connection.

The task that the `networking task source` places on the `task queue` once fetching for such a resource (with the correct `MIME type`) has completed must cause the user agent to reestablish the connection in parallel. This applies whether the connection is closed gracefully or unexpectedly (but does not apply when fetching is canceled by the user agent, e.g., in response to `window.stop()`, since in those cases the final task is actually discarded). It doesn't apply for the error conditions listed below except where explicitly specified.

HTTP 200 OK responses that have a `Content-Type` specifying an unsupported type, or that have no `Content-Type` at all, must cause the user agent to fail the connection.

Network errors that prevents the connection from being established in the first place (e.g. DNS errors), should cause the user agent to reestablish the connection in parallel, unless the user agent knows that to be futile, in which case the user agent may fail the connection.

Any other HTTP response code not listed here, as well as the cancelation of the fetch algorithm by the user agent (e.g. in response to `window.stop()` or the user canceling the network connection manually) must cause the user agent to fail the connection.

When a user agent is to announce the connection, the user agent must queue a task which, if the `readyState` attribute is set to a value other than `CLOSED`, sets the `readyState` attribute to `OPEN` and fires an event named `open` at the `EventSource` object.

When a user agent is to reestablish the connection, the user agent must run the following steps. These steps are run in parallel, not as part of a task. (The tasks that it queues, of course, are run like normal tasks and not themselves in parallel.)

1. Queue a task to run the following steps:
   1. If the `readyState` attribute is set to `CLOSED`, abort the task.
   2. Set the `readyState` attribute to `CONNECTING`.
   3. Fire an event named `error` at the `EventSource` object.

2. Wait a delay equal to the reconnection time of the event source.

3. Optionally, wait some more. In particular, if the previous attempt failed, then user agents might introduce an exponential backoff delay to avoid overloading a potentially already overloaded server. Alternatively, if the operating system has reported that there is no network connectivity, user agents might wait for the operating system to announce that the network connection has returned before retrying.

4. Wait until the aforementioned task has run, if it has not yet run.
5. Queue a task to run the following steps:

1. If the `EventSource` object's `readyState` attribute is not set to `CONNECTING`, return.

2. Let `request` be the `EventSource` object's `request`.

3. If the `EventSource` object's `last event ID string` is not the empty string, set `Last-Event-ID` as UTF-8, in `request`'s header list.

4. Fetch `request` and process the response obtained in this fashion, if any, as described earlier in this section.

When a user agent is to fail the connection, the user agent must queue a task which, if the `readyState` attribute is set to a value other than `CLOSED`, sets the `readyState` attribute to `CLOSED` and fires an event named `error` at the `EventSource` object. Once the user agent has failed the connection, it does not attempt to reconnect!

The task source for any tasks that are queued by `EventSource` objects is the remote event task source.

9.2.4 Parsing an event stream

This event stream format's MIME type is `text/event-stream`.

The event stream format is as described by the stream production of the following ABNF, the character set for which is Unicode.

```
stream = [ bom ] *event
event = *( comment / field ) end-of-line
comment = colon *any-char end-of-line
field = 1*name-char [ colon [ space ] *any-char ] end-of-line
end-of-line = ( cr lf / cr / lf )

; characters
lf = %x000A ; U+00A LINE FEED (LF)
cr = %x000D ; U+00D CARRIAGE RETURN (CR)
space = %x0020 ; U+002 SPACE
colon = %x003A ; U+003A COLON (:)
bom = %x0FFFE %x0EFF ; BYTE ORDER MARK
name-char = %x000-000 / %x000-00C / %x000-003E / %x003F-10FFFF
    ; a scalar value other than U+00A LINE FEED (LF), U+00D CARRIAGE RETURN (CR), or
U+003A COLON (:)
any-char = %x000-000 / %x000-00C / %x000-10FFFF
    ; a scalar value other than U+00A LINE FEED (LF) or U+00D CARRIAGE RETURN (CR)
```

Event streams in this format must always be encoded as UTF-8.

Lines must be separated by either a U+00D CARRIAGE RETURN U+00A LINE FEED (CRLF) character pair, a single U+00A LINE FEED (LF) character, or a single U+00D CARRIAGE RETURN (CR) character.

Since connections established to remote servers for such resources are expected to be long-lived, UAs should ensure that appropriate buffering is used. In particular, while line buffering with lines are defined to end with a single U+00A LINE FEED (LF) character is safe, block buffering or line buffering with different expected line endings can cause delays in event dispatch.

9.2.5 Interpreting an event stream

Streams must be decoded using the UTF-8 decode algorithm.

Note

The UTF-8 decode algorithm strips one leading UTF-8 Byte Order Mark (BOM), if any.
The stream must then be parsed by reading everything line by line, with a U+000D CARRIAGE RETURN U+000A LINE FEED (CRLF) character pair, a single U+000A LINE FEED (LF) character not preceded by a U+000D CARRIAGE RETURN (CR) character, and a single U+000D CARRIAGE RETURN (CR) character not followed by a U+000A LINE FEED (LF) character being the ways in which a line can end.

When a stream is parsed, a data buffer, an event type buffer, and a last event ID buffer must be associated with it. They must be initialized to the empty string.

Lines must be processed, in the order they are received, as follows:

- **If the line is empty (a blank line)**
  Dispatch the event\(^\text{987}\), as defined below.

- **If the line starts with a U+003A COLON character (:)**
  Ignore the line.

- **If the line contains a U+003A COLON character (:)**
  Collect the characters on the line before the first U+003A COLON character (:), and let field be that string.
  Collect the characters on the line after the first U+003A COLON character (:), and let value be that string. If value starts with a U+0020 SPACE character, remove it from value.
  Process the field\(^\text{987}\) using the steps described below, using field as the field name and value as the field value.

- **Otherwise, the string is not empty but does not contain a U+003A COLON character (:)**
  Process the field\(^\text{987}\) using the steps described below, using the whole line as the field name, and the empty string as the field value.

Once the end of the file is reached, any pending data must be discarded. (If the file ends in the middle of an event, before the final empty line, the incomplete event is not dispatched.)

The steps to process the field given a field name and a field value depend on the field name, as given in the following list. Field names must be compared literally, with no case folding performed.

- **If the field name is "event"**
  Set the event type buffer to field value.

- **If the field name is "data"**
  Append the field value to the data buffer, then append a single U+000A LINE FEED (LF) character to the data buffer.

- **If the field name is "id"**
  If the field value does not contain U+0000 NULL, then set the last event ID buffer to the field value. Otherwise, ignore the field.

- **If the field name is "retry"**
  If the field value consists of only ASCII digits, then interpret the field value as an integer in base ten, and set the event stream's reconnection time\(^\text{983}\) to that integer. Otherwise, ignore the field.

- **Otherwise**
  The field is ignored.

When the user agent is required to dispatch the event, the user agent must process the data buffer, the event type buffer, and the last event ID buffer using steps appropriate for the user agent.

For web browsers, the appropriate steps to dispatch the event\(^\text{987}\) are as follows:

1. Set the last event ID string\(^\text{983}\) of the event source to the value of the last event ID buffer. The buffer does not get reset, so the last event ID string\(^\text{983}\) of the event source remains set to this value until the next time it is set by the server.
2. If the data buffer is an empty string, set the data buffer and the event type buffer to the empty string and return.
3. If the data buffer's last character is a U+000A LINE FEED (LF) character, then remove the last character from the data buffer.
4. Let event be the result of creating an event using MessageEvent\(^\text{981}\), in the relevant Realm\(^\text{910}\) of the EventSource\(^\text{983}\) object.
5. Initialize event's type attribute to message,\(^\text{1266}\) its data attribute to data, its origin attribute to the serialization of...
the origin of the event stream's final URL (i.e., the URL after redirects), and its `lastEventId` attribute to the last event ID string of the event source.

6. If the event type buffer has a value other than the empty string, change the type of the newly created event to equal the value of the event type buffer.

7. Set the data buffer and the event type buffer to the empty string.

8. Queue a task which, if the readyState attribute is set to a value other than `CLOSED`, dispatches the newly created event at the EventSource object.

Note

If an event doesn't have an "id" field, but an earlier event did set the event source's last event ID string, then the event's lastEventId field will be set to the value of whatever the last seen "id" field was.

For other user agents, the appropriate steps to dispatch the event are implementation dependent, but at a minimum they must set the data and event type buffers to the empty string before returning.

Example

The following event stream, once followed by a blank line:

```
data: YHOO
 data: +2
 data: 10
```

...would cause an event message with the interface MessageEvent to be dispatched on the EventSource object. The event's data attribute would contain the string "YHOO
+2
10" (where "\n" represents a newline).

This could be used as follows:

```
var stocks = new EventSource("https://stocks.example.com/ticker.php");
stocks.onmessage = function (event) {
    var data = event.data.split('\n');
    updateStocks(data[0], data[1], data[2]);
};
```

...where updateStocks() is a function defined as:

```
function updateStocks(symbol, delta, value) { ... }
```

...or some such.

Example

The following stream contains four blocks. The first block has just a comment, and will fire nothing. The second block has two fields with names "data" and "id" respectively: an event will be fired for this block, with the data "first event", and will then set the last event ID to "1" so that if the connection died between this block and the next, the server would be sent a `Last-Event-ID` header with the value "1". The third block fires an event with data "second event", and also has an "id" field, this time with no value, which resets the last event ID to the empty string (meaning no `Last-Event-ID` header will now be sent in the event of a reconnection being attempted). Finally, the last block just fires an event with the data "third event" (with a single leading space character). Note that the last still has to end with a blank line, the end of the stream is not enough to trigger the dispatch of the last event.

: test stream

data: first event
 id: 1

data:second event
 id

data: third event
Legacy proxy servers are known to, in certain cases, drop HTTP connections after a short timeout. To protect against such proxy servers, authors can include a comment line (one starting with a `:` character) every 15 seconds or so.

Authors wishing to relate event source connections to each other or to specific documents previously served might find that relying on IP addresses doesn’t work, as individual clients can have multiple IP addresses (due to having multiple proxy servers) and individual IP addresses can have multiple clients (due to sharing a proxy server). It is better to include a unique identifier in the document when it is served and then pass that identifier as part of the URL when the connection is established.

Authors are also cautioned that HTTP chunking can have unexpected negative effects on the reliability of this protocol, in particular if the chunking is done by a different layer unaware of the timing requirements. If this is a problem, chunking can be disabled for serving event streams.

Clients that support HTTP’s per-server connection limitation might run into trouble when opening multiple pages from a site if each page has an `EventSource` to the same domain. Authors can avoid this using the relatively complex mechanism of using unique domain names per connection, or by allowing the user to enable or disable the `EventSource` functionality on a per-page basis, or by sharing a single `EventSource` object using a `shared worker`.

### 9.2.6 Authoring notes

Legacy proxy servers are known to, in certain cases, drop HTTP connections after a short timeout. To protect against such proxy servers, authors can include a comment line (one starting with a `:` character) every 15 seconds or so.

Authors wishing to relate event source connections to each other or to specific documents previously served might find that relying on IP addresses doesn’t work, as individual clients can have multiple IP addresses (due to having multiple proxy servers) and individual IP addresses can have multiple clients (due to sharing a proxy server). It is better to include a unique identifier in the document when it is served and then pass that identifier as part of the URL when the connection is established.

Authors are also cautioned that HTTP chunking can have unexpected negative effects on the reliability of this protocol, in particular if the chunking is done by a different layer unaware of the timing requirements. If this is a problem, chunking can be disabled for serving event streams.

Clients that support HTTP’s per-server connection limitation might run into trouble when opening multiple pages from a site if each page has an `EventSource` to the same domain. Authors can avoid this using the relatively complex mechanism of using unique domain names per connection, or by allowing the user to enable or disable the `EventSource` functionality on a per-page basis, or by sharing a single `EventSource` object using a `shared worker`.

### 9.2.7 Connectionless push and other features

User agents running in controlled environments, e.g. browsers on mobile handsets tied to specific carriers, may offload the management of the connection to a proxy on the network. In such a situation, the user agent for the purposes of conformance is considered to include both the handset software and the network proxy.

For example, a browser on a mobile device, after having established a connection, might detect that it is on a supporting network and request that a proxy server on the network take over the management of the connection. The timeline for such a situation might be as follows:

1. Browser connects to a remote HTTP server and requests the resource specified by the author in the `EventSource` constructor.
2. The server sends occasional messages.
3. In between two messages, the browser detects that it is idle except for the network activity involved in keeping the TCP connection alive, and decides to switch to sleep mode to save power.

4. The browser disconnects from the server.

5. The browser contacts a service on the network, and requests that the service, a "push proxy", maintain the connection instead.

6. The "push proxy" service contacts the remote HTTP server and requests the resource specified by the author in the EventSource constructor (possibly including a `Last-Event-ID` HTTP header, etc).

7. The browser allows the mobile device to go to sleep.

8. The server sends another message.

9. The "push proxy" service uses a technology such as OMA push to convey the event to the mobile device, which wakes only enough to process the event and then returns to sleep.

This can reduce the total data usage, and can therefore result in considerable power savings.

As well as implementing the existing API and `text/event-stream` wire format as defined by this specification and in more distributed ways as described above, formats of event framing defined by other applicable specifications may be supported. This specification does not define how they are to be parsed or processed.

### 9.2.8 Garbage collection

While an EventSource object's `readyState` is `CONNECTING`, and the object has one or more event listeners registered for `open`, `message`, or `error` events, there must be a strong reference from the Window or WorkerGlobalScope object that the EventSource object's constructor was invoked from to the EventSource object itself.

While an EventSource object's `readyState` is `OPEN`, and the object has one or more event listeners registered for `message` or `error` events, there must be a strong reference from the Window or WorkerGlobalScope object that the EventSource object's constructor was invoked from to that EventSource object.

While there is a task queued by an EventSource object on the remote event task source, there must be a strong reference from the Window or WorkerGlobalScope object that the EventSource object's constructor was invoked from to that EventSource object.

If a user agent is to forcibly close an EventSource object (this happens when a Document object goes away permanently), the user agent must abort any instances of the fetch algorithm started for this EventSource object, and must set the `readyState` attribute to `CLOSED`.

If an EventSource object is garbage collected while its connection is still open, the user agent must abort any instance of the fetch algorithm opened by this EventSource object.

### 9.2.9 Implementation advice

This section is non-normative.

User agents are strongly urged to provide detailed diagnostic information about EventSource objects and their related network connections in their development consoles, to aid authors in debugging code using this API.

For example, a user agent could have a panel displaying all the EventSource objects a page has created, each listing the constructor's arguments, whether there was a network error, what the CORS status of the connection is and what headers were sent by the client and received from the server to lead to that status, the messages that were received and how they were parsed, and so forth.

Implementations are especially encouraged to report detailed information to their development consoles whenever an error event
is fired, since little to no information can be made available in the events themselves.

9.3 Web sockets

9.3.1 Introduction

This section is non-normative.

To enable web applications to maintain bidirectional communications with server-side processes, this specification introduces the WebSocket interface.

Note

This interface does not allow for raw access to the underlying network. For example, this interface could not be used to implement an IRC client without proxying messages through a custom server.

9.3.2 The WebSocket interface

IDL

```idl
enum BinaryType { "blob", "arraybuffer" }; [Exposed=(Window,Worker)]
interface WebSocket : EventTarget {
    constructor (USVString url, optional (DOMString or sequence<DOMString>) protocols = []);
    readonly attribute USVString url;

    // ready state
    const unsigned short CONNECTING = 0;
    const unsigned short OPEN = 1;
    const unsigned short CLOSING = 2;
    const unsigned short CLOSED = 3;
    readonly attribute unsigned short readyState;
    readonly attribute unsigned long long bufferedAmount;

    // networking
    attribute EventHandler onopen;
    attribute EventHandler onerror;
    attribute EventHandler onclose;
    readonly attribute DOMString extensions;
    readonly attribute DOMString protocol;
    undefined close(optional [Clamp] unsigned short code, optional USVString reason);

    // messaging
    attribute EventHandler onmessage;
    attribute BinaryType binaryType;
    undefined send(USVString data);
    undefined send(Blob data);
    undefined send(ArrayBuffer data);
    undefined send(ArrayBufferView data);
};
```

Each WebSocket object has an associated url (a URL record).

For web developers (non-normative)
The WebSocket constructor, when invoked, must run these steps:

1. Let urlRecord be the result of applying the URL parser to url.
2. If urlRecord is failure, then throw a "SyntaxError" DOMException.
3. If urlRecord's scheme is not "ws" or "wss", then throw a "SyntaxError" DOMException.
4. If urlRecord's fragment is non-null, then throw a "SyntaxError" DOMException.
5. If protocols is a string, set protocols to a sequence consisting of just that string.
6. If any of the values in protocols occur more than once or otherwise fail to match the requirements for elements that comprise the value of Sec-WebSocket-Protocol fields as defined by The WebSocket protocol, then throw a "SyntaxError" DOMException.
7. Run this step in parallel:
   1. Establish a WebSocket connection given urlRecord, protocols, and the entry settings object, then:

```javascript
socket = new WebSocket(url [, protocols ])
```

Creates a new WebSocket object, immediately establishing the associated WebSocket connection. 

url is a string giving the URL over which the connection is established. Only "ws" or "wss" schemes are allowed; others will cause a "SyntaxError" DOMException. URLs with fragments will also cause such an exception.

protocols is either a string or an array of strings. If it is a string, it is equivalent to an array consisting of just that string; if it is omitted, it is equivalent to the empty array. Each string in the array is a subprotocol name. The connection will only be established if the server reports that it has selected one of these subprotocols. The subprotocol names have to match the requirements for elements that comprise the value of Sec-WebSocket-Protocol fields as defined by The WebSocket protocol.

```javascript
socket . send(data)
```

Transmits data using the WebSocket connection. data can be a string, a Blob, an ArrayBuffer, or an ArrayBufferView.

```javascript
socket . close([code] [, reason ])
```

Closes the WebSocket connection, optionally using code as the the WebSocket connection close code and reason as the the WebSocket connection close reason.

```javascript
socket . url
```

Returns the URL that was used to establish the WebSocket connection.

```javascript
socket . readyState
```

Returns the state of the WebSocket object's connection. It can have the values described below.

```javascript
socket . bufferedAmount
```

Returns the number of bytes of application data (UTF-8 text and binary data) that have been queued using send() but not yet been transmitted to the network.

If the WebSocket connection is closed, this attribute's value will only increase with each call to the send() method. (The number does not reset to zero once the connection closes.)

```javascript
socket . extensions
```

Returns the extensions selected by the server, if any.

```javascript
socket . protocol
```

Returns the subprotocol selected by the server, if any. It can be used in conjunction with the array form of the constructor's second argument to perform subprotocol negotiation.

```javascript
socket . binaryType [= value ]
```

Returns a string that indicates how binary data from the WebSocket object is exposed to scripts:

- "blob": Binary data is returned in Blob form.
- "arraybuffer": Binary data is returned in ArrayBuffer form.

Can be set, to change how binary data is returned. The default is "blob".
8. Return a new `WebSocket` object whose `url` is `urlRecord`.

The `url` attribute's getter must return this `WebSocket` object's `url`, serialized.

The `readyState` attribute represents the state of the connection. It can have the following values:

**CONNECTING (numeric value 0)**

The connection has not yet been established.

**OPEN (numeric value 1)**

The WebSocket connection is established and communication is possible.

**CLOSING (numeric value 2)**

The connection is going through the closing handshake, or the `close()` method has been invoked.

**CLOSED (numeric value 3)**

The connection has been closed or could not be opened.

When the object is created its `readyState` must be set to `CONNECTING` (0).

The `extensions` attribute must initially return the empty string. After the WebSocket connection is established, its value might change, as defined below.

The `protocol` attribute must initially return the empty string. After the WebSocket connection is established, its value might change, as defined below.

The `close(code, reason)` method, when invoked, must run these steps:

1. If `code` is present, but is neither an integer equal to 1000 nor an integer in the range 3000 to 4999, inclusive, throw an "InvalidAccessError" DOMException.
2. If `reason` is present, then run these substeps:
   1. Let `reasonBytes` be the result of encoding `reason`.
   2. If `reasonBytes` is longer than 123 bytes, then throw a "SyntaxError" DOMException.
3. Run the first matching steps from the following list:
   
   ➔ If the `readyState` attribute is in the `CLOSING` (2) or `CLOSED` (3) state
      
      Do nothing.

   ✔ Note
   
   The connection is already closing or is already closed. If it has not already, a `close` event will eventually fire as described below.

   ➔ If the WebSocket connection is not yet `established`, [WSP]
      
      Fail the WebSocket connection and set the `readyState` attribute's value to `CLOSING` (2). [WSP]

   ✔ Note
   
   The fail the WebSocket connection algorithm invokes the close the WebSocket connection algorithm, which then establishes that the WebSocket connection is closed, which fires the `close` event as described below.

   ➔ If the WebSocket closing handshake has not yet been `started`, [WSP]
      
      Start the WebSocket closing handshake and set the `readyState` attribute's value to `CLOSING` (2). [WSP]

   If neither `code` nor `reason` is present, the WebSocket Close message must not have a body.
If code is present, then the status code to use in the WebSocket Close message must be the integer given by close. [WSP]1278

If reason is also present, then reasonBytes must be provided in the Close message after the status code. [WSP]1278

Note
The start the WebSocket closing handshake[p63] algorithm eventually invokes the close the WebSocket connection[p63] algorithm, which then establishes that the WebSocket connection is closed[p63], which fires the close[p1266] event as described below[min].

Otherwise
Set the readyState[993] attribute's value to CLOSING[993] (2).

Note
The WebSoc ket closing handshake is started[p63], and will eventually invoke the close the WebSocket connection[p63] algorithm, which will establish that the WebSocket connection is closed[p63], and thus the close[p1266] event will fire, as described below[996].

Note
The close[p993] method does not discard previously sent messages before starting the WebSocket closing handshake — even if, in practice, the user agent is still busy sending those messages, the handshake will only start after the messages are sent.

The bufferedAmount attribute must return the number of bytes of application data (UTF-8 text and binary data) that have been queued using send()[995] but that, as of the last time the event loop[993] reached step 1[993], had not yet been transmitted to the network. (This thus includes any text sent during the execution of the current task, regardless of whether the user agent is able to transmit text in the background in parallel[p93] with script execution.) This does not include framing overhead incurred by the protocol, or buffering done by the operating system or network hardware.

Example
In this simple example, the bufferedAmount[994] attribute is used to ensure that updates are sent either at the rate of one update every 50ms, if the network can handle that rate, or at whatever rate the network can handle, if that is too fast.

```javascript
var socket = new WebSocket('ws://game.example.com:12010/updates');
socket.onopen = function () {
  setInterval(function () {
    if (socket.bufferedAmount == 0)
      socket.send(getUpdateData());
  }, 50);
};
```

The bufferedAmount[994] attribute can also be used to saturate the network without sending the data at a higher rate than the network can handle, though this requires more careful monitoring of the value of the attribute over time.

When a WebSocket[991] object is created, its binaryType IDL attribute must be set to the string "blob[996]". On getting, it must return the last value it was set to. On setting, the user agent must set the IDL attribute to the new value.

Note
User agents can use the binaryType[994] attribute as a hint for how to handle incoming binary data: if the attribute is set to "blob[996]", it is safe to spool it to disk, and if it is set to "arraybuffer[996]", it is likely more efficient to keep the data in memory. Naturally, user agents are encouraged to use more subtle heuristics to decide whether to keep incoming data in memory or not, e.g. based on how big the data is or how common it is for a script to change the attribute at the last minute. This latter aspect is important in particular because it is quite possible for the attribute to be changed after the user agent has received the data but
The `send(data)` method transmits data using the connection. If the `readyState` attribute is `CONNECTING`, it must throw an "InvalidStateError" `DOMException`. Otherwise, the user agent must run the appropriate set of steps from the following list:

If the argument is a string

If the WebSocket connection is established and the WebSocket closing handshake has not yet started, the user agent must send a WebSocket Message comprised of the data argument using a text frame opcode; if the data cannot be sent, e.g. because it would need to be buffered but the buffer is full, the user agent must flag the WebSocket as full and then close the WebSocket connection. Any invocation of this method with a string argument that does not throw an exception must increase the `bufferedAmount` attribute by the number of bytes needed to express the argument as UTF-8.

If the argument is a `Blob` object

If the WebSocket connection is established and the WebSocket closing handshake has not yet started, the user agent must send a WebSocket Message comprised of data using a binary frame opcode; if the data cannot be sent, e.g. because it would need to be buffered but the buffer is full, the user agent must flag the WebSocket as full and then close the WebSocket connection. The data to be sent is the raw data represented by the `Blob` object. Any invocation of this method with a `Blob` argument that does not throw an exception must increase the `bufferedAmount` attribute by the size of the `Blob` object's raw data, in bytes.

If the argument is an `ArrayBuffer` object

If the WebSocket connection is established and the WebSocket closing handshake has not yet started, the user agent must send a WebSocket Message comprised of data using a binary frame opcode; if the data cannot be sent, e.g. because it would need to be buffered but the buffer is full, the user agent must flag the WebSocket as full and then close the WebSocket connection. The data to be sent is the data stored in the buffer described by the `ArrayBuffer` object. Any invocation of this method with an `ArrayBuffer` argument that does not throw an exception must increase the `bufferedAmount` attribute by the length of the `ArrayBuffer` in bytes.

If the argument is an object that matches the `ArrayBufferView` type definition

If the WebSocket connection is established and the WebSocket closing handshake has not yet started, the user agent must send a WebSocket Message comprised of data using a binary frame opcode; if the data cannot be sent, e.g. because it would need to be buffered but the buffer is full, the user agent must flag the WebSocket as full and then close the WebSocket connection. The data to be sent is the data stored in the section of the buffer described by the `ArrayBuffer` object that data references. Any invocation of this method with this kind of argument that does not throw an exception must increase the `bufferedAmount` attribute by the length of data's buffer in bytes.

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 `WebSocket` interface:

<table>
<thead>
<tr>
<th>Event handler</th>
<th>Event handler event type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>onopen</code></td>
<td><code>open</code></td>
</tr>
<tr>
<td><code>onmessage</code></td>
<td><code>message</code></td>
</tr>
<tr>
<td><code>onerror</code></td>
<td><code>error</code></td>
</tr>
<tr>
<td><code>onclose</code></td>
<td><code>close</code></td>
</tr>
</tbody>
</table>

9.3.3 Feedback from the protocol

When the WebSocket connection is established, the user agent must queue a task to run these steps:

1. Change the `readyState` attribute's value to `OPEN` (1).
2. Change the `extensions` attribute's value to the `extensions in use`, if it is not the null value.
3. Change the `protocol` attribute's value to the `subprotocol in use`, if it is not the null value.
4. Fire an event named `open` at the `WebSocket` object.
When a WebSocket message has been received with type and data, the user agent must queue a task to follow these steps:  

1. If the readyState attribute's value is not OPEN, then return.
2. Let dataForEvent be determined by switching on type and binaryType:
   - If type indicates that the data is Text, a new DOMString containing data.
   - If type indicates that the data is Binary and binaryType is "blob", a new Blob object, created in the relevant Realm of the WebSocket object, that represents data as its raw data.
   - If type indicates that the data is Binary and binaryType is "arraybuffer", a new ArrayBuffer object, created in the relevant Realm of the WebSocket object, whose contents are data.
3. Fire an event named message at the WebSocket object, using MessageEvent, with the origin attribute initialized to the serialization of the WebSocket object's url's origin, and the data attribute initialized to dataForEvent.

When the WebSocket closing handshake is started, the user agent must queue a task to change the readyState attribute's value to CLOSING. (If the close() method was called, the readyState attribute's value will already be set to CLOSING when this task runs.)

When the WebSocket connection is closed, possibly cleanly, the user agent must queue a task to run the following substeps:

1. Change the readyState attribute's value to CLOSED (3).
2. If the user agent was required to fail the WebSocket connection, or if the the WebSocket connection was closed after being flagged as full, fire an event named error at the WebSocket object.
3. Fire an event named close at the WebSocket object, using CloseEvent, with the wasClean attribute initialized to true if the connection closed cleanly and false otherwise, the code attribute initialized to the WebSocket connection close code, and the reason attribute initialized to the result of applying UTF-8 decode without BOM to the WebSocket connection close code.
The task source for all tasks queued in this section is the WebSocket task source.

### 9.3.4 Ping and Pong frames

The WebSocket protocol defines Ping and Pong frames that can be used for keep-alive, heart-beats, network status probing, latency instrumentation, and so forth. These are not currently exposed in the API.

User agents may send ping and unsolicited pong frames as desired, for example in an attempt to maintain local network NAT mappings, to detect failed connections, or to display latency metrics to the user. User agents must not use pings or unsolicited pongs to aid the server; it is assumed that servers will solicit pongs whenever appropriate for the server's needs.

### 9.3.5 The CloseEvent interface

WebSocket objects use the CloseEvent interface for their close events:

```idl
[Exposed=(Window, Worker)]
interface CloseEvent : Event {
    constructor(DOMString type, optional CloseEventInit eventInitDict = {});

    readonly attribute boolean wasClean;
    readonly attribute unsigned short code;
    readonly attribute USVString reason;
};
```

In all of these cases, the WebSocket connection close code would be 1006, as required by WebSocket Protocol. Allowing a script to distinguish these cases would allow a script to probe the user's local network in preparation for an attack.

**Warning!**

User agents must not convey any failure information to scripts in a way that would allow a script to distinguish the following situations:

- A server whose host name could not be resolved.
- A server to which packets could not successfully be routed.
- A server that refused the connection on the specified port.
- A server that failed to correctly perform a TLS handshake (e.g., the server certificate can't be verified).
- A server that did not complete the opening handshake (e.g. because it was not a WebSocket server).
- A WebSocket server that sent a correct opening handshake, but that specified options that caused the client to drop the connection (e.g. the server specified a subprotocol that the client did not offer).
- A WebSocket server that abruptly closed the connection after successfully completing the opening handshake.

In particular, this means the code 1015 is not used by the user agent (unless the server erroneously uses it in its close frame, of course).
boolean wasClean = false;
unsigned short code = 0;
USVString reason = "";
};

For web developers (non-normative)

**event wasClean**

Returns true if the connection closed cleanly; false otherwise.

**event code**

Returns the WebSocket connection close code provided by the server.

**event reason**

Returns the WebSocket connection close reason provided by the server.

The `wasClean` attribute must return the value it was initialized to. It represents whether the connection closed cleanly or not.

The `code` attribute must return the value it was initialized to. It represents the WebSocket connection close code provided by the server.

The `reason` attribute must return the value it was initialized to. It represents the WebSocket connection close reason provided by the server.

9.3.6 Garbage collection

A WebSocket object whose `readyState` attribute's value was set to `CONNECTING` (0) as of the last time the `event loop` reached step 1 must not be garbage collected if there are any event listeners registered for `open` events, `message` events, `error` events, or `close` events.

A WebSocket object whose `readyState` attribute's value was set to `OPEN` (1) as of the last time the `event loop` reached step 1 must not be garbage collected if there are any event listeners registered for `message` events, `error` or `close` events.

A WebSocket object whose `readyState` attribute's value was set to `CLOSING` (2) as of the last time the `event loop` reached step 1 must not be garbage collected if there are any event listeners registered for `error` or `close` events.

A WebSocket object with an established connection that has data queued to be transmitted to the network must not be garbage collected.

If a WebSocket object is garbage collected while its connection is still open, the user agent must start the WebSocket closing handshake, with no status code for the Close message.

If a user agent is to make disappear a WebSocket object (this happens when a `Document` object goes away), the user agent must follow the first appropriate set of steps from the following list:

- If the WebSocket connection is not yet established, fail the WebSocket connection.

- If the WebSocket closing handshake has not yet been started, start the WebSocket closing handshake, with the status code to use in the WebSocket Close message being 1001.

- Otherwise, do nothing.
9.4 Cross-document messaging

Web browsers, for security and privacy reasons, prevent documents in different domains from affecting each other; that is, cross-site scripting is disallowed.

While this is an important security feature, it prevents pages from different domains from communicating even when those pages are not hostile. This section introduces a messaging system that allows documents to communicate with each other regardless of their source domain, in a way designed to not enable cross-site scripting attacks.

Note
The `postMessage()` API can be used as a tracking vector.

9.4.1 Introduction

This section is non-normative.

Example

For example, if document A contains an `iframe` element that contains document B, and script in document A calls `postMessage()` on the `Window` object of document B, then a message event will be fired on that object, marked as originating from the `Window` of document A. The script in document A might look like:

```javascript
var o = document.getElementsByTagName('iframe')[0];
o.contentWindow.postMessage('Hello world', 'https://b.example.org/');
```

To register an event handler for incoming events, the script would use `addEventListener()` (or similar mechanisms). For example, the script in document B might look like:

```javascript
window.addEventListener('message', receiver, false);
function receiver(e) {
    if (e.origin == 'https://example.com') {
        if (e.data == 'Hello world') {
            e.source.postMessage('Hello', e.origin);
        } else {
            alert(e.data);
        }
    }
}
```

This script first checks the domain is the expected domain, and then looks at the message, which it either displays to the user, or responds to by sending a message back to the document which sent the message in the first place.

9.4.2 Security

9.4.2.1 Authors

⚠️ Warning!
Use of this API requires extra care to protect users from hostile entities abusing a site for their own purposes.

Authors should check the `origin` attribute to ensure that messages are only accepted from domains that they expect to receive messages from. Otherwise, bugs in the author's message handling code could be exploited by hostile sites.

Furthermore, even after checking the `origin` attribute, authors should also check that the data in question is of the expected format. Otherwise, if the source of the event has been attacked using a cross-site scripting flaw, further unchecked processing of information sent using the `postMessage()` method could result in the attack being propagated into the receiver.

Authors should not use the wildcard keyword (*) in the `targetOrigin` argument in messages that contain any confidential information, as
otherwise there is no way to guarantee that the message is only delivered to the recipient to which it was intended.

Authors who accept messages from any origin are encouraged to consider the risks of a denial-of-service attack. An attacker could send a high volume of messages; if the receiving page performs expensive computation or causes network traffic to be sent for each such message, the attacker’s message could be multiplied into a denial-of-service attack. Authors are encouraged to employ rate limiting (only accepting a certain number of messages per minute) to make such attacks impractical.

9.4.2.2 User agents

The integrity of this API is based on the inability for scripts of one origin to post arbitrary events (using dispatchEvent() or otherwise) to objects in other origins (those that are not the same).

Note
Implementors are urged to take extra care in the implementation of this feature. It allows authors to transmit information from one domain to another domain, which is normally disallowed for security reasons. It also requires that UAs be careful to allow access to certain properties but not others.

User agents are also encouraged to consider rate-limiting message traffic between different origins, to protect naïve sites from denial-of-service attacks.

9.4.3 Posting messages

For web developers (non-normative)

window.postMessage(message [, options ])

Posts a message to the given window. Messages can be structured objects, e.g. nested objects and arrays, can contain JavaScript values (strings, numbers, Date objects, etc), and can contain certain data objects such as File, Blob, FileList, and ArrayBuffer objects.

Objects listed in the transfer member of options are transferred, not just cloned, meaning that they are no longer usable on the sending side.

A target origin can be specified using the targetOrigin member of options. If not provided, it defaults to "/". This default restricts the message to same-origin targets only.

If the origin of the target window doesn’t match the given target origin, the message is discarded, to avoid information leakage. To send the message to the target regardless of origin, set the target origin to "/*

Throws a "DataCloneError" DOMException if transfer array contains duplicate objects or if message could not be cloned.

window.postMessage(message, targetOrigin [, transfer ])

This is an alternate version of postMessage() where the target origin is specified as a parameter. Calling window.postMessage(message, target, transfer) is equivalent to window.postMessage(message, {targetOrigin, transfer}).

Note

When posting a message to a Window of a browsing context that has just been navigated to a new Document is likely to result in the message not receiving its intended recipient: the scripts in the target browsing context have to have had time to set up listeners for the messages. Thus, for instance, in situations where a message is to be sent to the Window of newly created child iframes, authors are advised to have the child Document post a message to their parent announcing their readiness to receive messages, and for the parent to wait for this message before beginning posting messages.

The window post message steps, given a targetWindow, message, and options, are as follows:

1. Let targetRealm be targetWindow’s Realm.
2. Let incumbentSettings be the incumbent settings object.

1000
3. Let targetOrigin be `options["targetOrigin"]`.

4. If `targetOrigin` is a single U+002F SOLIDUS character (/), then set `targetOrigin` to `incumbentSettings`'s `origin`.

5. Otherwise, if `targetOrigin` is not a single U+002A ASTERISK character (*), then:
   1. Let `parsedURL` be the result of running the URL parser on `targetOrigin`.
   2. If `parsedURL` is failure, then throw a "SyntaxError" DOMException.
   3. Set `targetOrigin` to `parsedURL`'s `origin`.

6. Let `transfer` be `options["transfer"]`.

7. Let `serializeWithTransferResult` be `StructuredSerializeWithTransfer(message, transfer)`. Rethrow any exceptions.

8. Queue a global task on the posted message task source given `targetWindow` to run the following steps:
   1. If the `targetOrigin` argument is not a single literal U+002A ASTERISK character (*) and `targetWindow`'s associated Document's `origin` is not same origin with `targetOrigin`, then return.
   2. Let `origin` be the serialization of `incumbentSettings`'s `origin`.
   3. Let `source` be the `WindowProxy` object corresponding to `incumbentSettings`'s `global object` (a `Window` object).
   4. Let `deserializeRecord` be `StructuredDeserializeWithTransfer(serializeWithTransferResult, targetRealm)`. If this throws an exception, catch it, fire an event named `messageerror` at `targetWindow`, using `MessageEvent`, with the `origin` attribute initialized to `origin` and the `source` attribute initialized to `source`, and then return.
   5. Let `messageClone` be `deserializeRecord.[[Deserialized]]`.
   6. Let `newPorts` be a new `frozen array` consisting of all `MessagePort` objects in `deserializeRecord.[[TransferredValues]]`, if any, maintaining their relative order.
   7. Fire an event named `message` at `targetWindow`, using `MessageEvent`, with the `origin` attribute initialized to `origin`, the `source` attribute initialized to `source`, the `data` attribute initialized to `messageClone`, and the `ports` attribute initialized to `newPorts`.

The `postMessage(message, options)` method, when invoked on a `Window` object, must run the following steps:

1. Let `targetWindow` be this `Window` object.
2. Run the window post message steps providing `targetWindow`, `message`, and `options`.

The `postMessage(message, targetOrigin, transfer)` method, when invoked on a `Window` object, must run the following steps:

1. Let `targetWindow` be this `Window` object.
2. Let `options` be `{{"targetOrigin" → targetOrigin, "transfer" → transfer }}`.
3. Run the window post message steps providing `targetWindow`, `message`, and `options`.

### 9.5 Channel messaging

#### 9.5.1 Introduction

This section is non-normative.

To enable independent pieces of code (e.g. running in different browsing contexts) to communicate directly, authors can use channel messaging.
Communication channels in this mechanism are implemented as two-ways pipes, with a port at each end. Messages sent in one port are delivered at the other port, and vice-versa. Messages are delivered as DOM events, without interrupting or blocking running tasks.

To create a connection (two "entangled" ports), the MessageChannel() constructor is called:

```javascript
var channel = new MessageChannel();
```

One of the ports is kept as the local port, and the other port is sent to the remote code, e.g. using `postMessage()`:

```javascript
otherWindow.postMessage('hello', 'https://example.com', [channel.port2]);
```

To send messages, the `postMessage()` method on the port is used:

```javascript
channel.port1.postMessage('hello');
```

To receive messages, one listens to `message` events:

```javascript
channel.port1.onmessage = handleMessage;
function handleMessage(event) {
    // message is in event.data
    // ...
}
```

Data sent on a port can be structured data; for example here an array of strings is passed on a `MessagePort`:

```javascript
port1.postMessage(['hello', 'world']);
```

This section is non-normative.

### 9.5.1.1 Examples

This section is non-normative.

**Example**

In this example, two JavaScript libraries are connected to each other using `MessagePort`s. This allows the libraries to later be hosted in different frames, or in `Worker` objects, without any change to the APIs.

```html
<script src="contacts.js"></script> <!-- exposes a contacts object -->
<script src="compose-mail.js"></script> <!-- exposes a composer object -->
<script>
    var channel = new MessageChannel();
    composer.addContactsProvider(channel.port1);
    contacts.registerConsumer(channel.port2);
</script>
```

Here's what the "addContactsProvider()" function's implementation could look like:

```javascript
function addContactsProvider(port) {
    port.onmessage = function (event) {
        switch (event.data.messageType) {
            case 'search-result':
                handleSearchResult(event.data.results);
                break;
            case 'search-done':
                handleSearchDone();
                break;
            case 'search-error':
                handleSearchError(event.data.message);
                break;
            // ...
        }
    }
}
```
This section is non-normative.

Ports can be viewed as a way to expose limited capabilities (in the object-capability model sense) to other actors in the system. This can either be a weak capability system, where the ports are merely used as a convenient model within a particular origin, or as a strong capability model, where they are provided by one origin \textit{provider} as the only mechanism by which another origin \textit{consumer} can effect change in or obtain information from \textit{provider}.

For example, consider a situation in which a social web site embeds in one iframe the user's email contacts provider (an address book site, from a second origin), and in a second iframe a game (from a third origin). The outer social site and the game in the second iframe cannot access anything inside the first iframe; together they can only:

- \textbf{Navigate} to a new URL, such as the same URL but with a different fragment, causing the Window's \texttt{hashchange} event.
- \textbf{Resize} the iframe, causing the Window's \texttt{resize} event.
- Send a message event to the Window in the iframe using the window.postMessage API.

The contacts provider can use these methods, most particularly the third one, to provide an API that can be accessed by other origins to manipulate the user's address book. For example, it could respond to a message "add-contact Guillaume Tell <tell@pomme.example.net>" by adding the given person and email address to the user's address book.

To avoid any site on the web being able to manipulate the user's contacts, the contacts provider might only allow certain trusted sites, such as the social site, to do this.

Now suppose the game wanted to add a contact to the user's address book, and that the social site was willing to allow it to do so on its behalf, essentially "sharing" the trust that the contacts provider had with the social site. There are several ways it could do this; most simply, it could just proxy messages between the game site and the contacts site. However, this solution has a number of difficulties: it requires the social site to either completely trust the game site not to abuse the privilege, or it requires that the social site verify each request to make sure it's not a request that it doesn't want to allow (such as adding multiple contacts, reading the contacts, or deleting them); it also requires some additional complexity if there's ever the possibility of multiple games simultaneously trying to interact with the contacts provider.

Using message channels and MessagePort objects, however, all of these problems can go away. When the game tells the social
site that it wants to add a contact, the social site can ask the contacts provider not for it to add a contact, but for the capability to add a single contact. The contacts provider then creates a pair of MessagePort objects, and sends one of them back to the social site, who forwards it on to the game. The game and the contacts provider then have a direct connection, and the contacts provider knows to only honor a single “add contact” request, nothing else. In other words, the game has been granted the capability to add a single contact.

9.5.1.3 Ports as the basis of abstracting out service implementations

This section is non-normative.

Continuing the example from the previous section, consider the contacts provider in particular. While an initial implementation might have simply used XMLHttpRequest objects in the service’s iframe, an evolution of the service might instead want to use a shared worker with a single WebSocket connection.

If the initial design used MessagePort objects to grant capabilities, or even just to allow multiple simultaneous independent sessions, the service implementation can switch from the XMLHttpRequests-in-each-iframe model to the shared-WebSocket model without changing the API at all: the ports on the service provider side can all be forwarded to the shared worker without it affecting the users of the API in the slightest.

9.5.2 Message channels

IDL

```idl
[Exposed=(Window,Worker)]
interface MessageChannel {
    constructor();

    readonly attribute MessagePort port1;
    readonly attribute MessagePort port2;
};
```

A MessageChannel object has an associated port 1 and an associated port 2, both MessagePort objects.

The new MessageChannel() constructor steps are:

1. Set this's port 1 to a new MessagePort in this's relevant Realm.
2. Set this's port 2 to a new MessagePort in this's relevant Realm.
3. Entangle this's port 1 and this's port 2.

The port1 getter steps are to return this's port 1.

The port2 getter steps are to return this's port 2.

9.5.3 Message ports

Each channel has two message ports. Data sent through one port is received by the other port, and vice versa.
IDL

```idl
[Exposed=(Window, Worker, AudioWorklet), Transferable]
interface MessagePort : EventTarget {
    undefined postMessage(any message, sequence<object> transfer);
    undefined postMessage(any message, optional PostMessageOptions options = {});
    undefined start();
    undefined close();

    // event handlers
    attribute EventHandler onmessage;
    attribute EventHandler onmessageerror;
};
dictionary PostMessageOptions {
    sequence<object> transfer = [];
};
```

For web developers (non-normative)

```javascript
port . postMessage(message [, transfer ] )
port . postMessage(message [, { transfer } ])
```

Posts a message through the channel. Objects listed in `transfer` are transferred, not just cloned, meaning that they are no longer usable on the sending side.

Throws a "DataCloneError" DOMException if `transfer` contains duplicate objects or `port`, or if `message` could not be cloned.

```javascript
port . start()
```

Begins dispatching messages received on the port.

```javascript
port . close()
```

Disconnects the port, so that it is no longer active.

Each `MessagePort` object can be entangled with another (a symmetric relationship). Each `MessagePort` object also has a task source called the port message queue, initially empty. A port message queue can be enabled or disabled, and is initially disabled. Once enabled, a port can never be disabled again (though messages in the queue can get moved to another queue or removed altogether, which has much the same effect). A `MessagePort` also has a has been shipped flag, which must initially be false.

When a port's port message queue is enabled, the event loop must use it as one of its task sources. When a port's relevant global object is a `Window`, all tasks, queued on its port message queue, must be associated with the port's relevant global object's associated Document.

Note

If the document is fully active, but the event listeners all have scripts whose settings objects specify responsible documents that are not fully active, then the messages will not be received unless and until the documents become fully active again.

Each event loop has a task source called the unshipped port message queue. This is a virtual task source, it must act as if it contained the tasks of each port message queue of each `MessagePort` whose has been shipped flag is false, whose port message queue is enabled, and whose relevant agent's event loop is that event loop, in the order in which they were added to their respective task source. When a task would be removed from the unshipped port message queue, it must instead be removed from its port message queue.

When a `MessagePort`'s has been shipped flag is false, its port message queue must be ignored for the purposes of the event loop. (The unshipped port message queue is used instead.)

Note

The has been shipped flag is set to true when a port, its twin, or the object it was cloned from, is or has been transferred. When a `MessagePort`'s has been shipped flag is true, its port message queue acts as a first-class task source, unaffected to any unshipped port message queue.

When the user agent is to entangle two `MessagePort` objects, it must run the following steps:
1. If one of the ports is already entangled, then disentangle it and the port that it was entangled with.

   **Note**
   
   *If those two previously entangled ports were the two ports of a `MessageChannel` object, then that `MessageChannel` object no longer represents an actual channel: the two ports in that object are no longer entangled.*

2. Associate the two ports to be entangled, so that they form the two parts of a new channel. (There is no `MessageChannel` object that represents this channel.)

Two ports A and B that have gone through this step are now said to be entangled; one is entangled to the other, and vice versa.

   **Note**
   
   *While this specification describes this process as instantaneous, implementations are more likely to implement it via message passing. As with all algorithms, the key is “merely” that the end result be indistinguishable, in a black-box sense, from the specification.*

`MessagePort` objects are transferable objects. Their transfer steps, given `value` and `dataHolder`, are:

1. Set `value`'s has been shipped flag to true.
2. Set `dataHolder`.[PortMessageQueue] to `value`'s port message queue.
3. If `value` is entangled with another port `remotePort`, then:
   1. Set `remotePort`'s has been shipped flag to true.
   2. Set `dataHolder`.[RemotePort] to `remotePort`.
4. Otherwise, set `dataHolder`.[RemotePort] to null.

Their transfer-receiving steps, given `dataHolder` and `value`, are:

1. Set `value`'s has been shipped flag to true.
2. Move all the tasks that are to fire message events in `dataHolder`.[PortMessageQueue] to the port message queue of `value`, if any, leaving `value`'s port message queue in its initial disabled state, and, if `value`'s relevant global object is a `Window`, associating the moved tasks with `value`'s relevant global object's associated Document.
3. If `dataHolder`.[RemotePort] is not null, then entangle `dataHolder`.[RemotePort] and `value`. (This will disentangle `dataHolder`.[RemotePort] from the original port that was transferred.)

The message port post message steps, given a `targetPort`, `message` and `options` are as follows:

1. Let `transfer` be options["transfer"].
2. If `transfer` contains this `MessagePort`, then throw a "DataCloneError" DOMException.
3. Let `doomed` be false.
4. If `targetPort` is not null and `transfer` contains `targetPort`, then set `doomed` to true and optionally report to a developer console that the target port was posted to itself, causing the communication channel to be lost.
5. Let `serializeWithTransferResult` be `StructuredSerializeWithTransfer(message, transfer)`. Rethrow any exceptions.
6. If `targetPort` is null, or if `doomed` is true, then return.
7. Add a `task` that runs the following steps to the port message queue of `targetPort`:
   1. Let `finalTargetPort` be the `MessagePort` in whose port message queue the task now finds itself.

   **Note**
   
   *This can be different from targetPort, if targetPort itself was transferred and thus all its tasks moved along with it.*
2. Let targetRealm be `finalTargetPort's relevant Realm`.

3. Let deserializeRecord be `StructuredDeserializeWithTransfer(serializeWithTransferResult, targetRealm)`. If this throws an exception, catch it, fire an event named `messageerror` at `finalTargetPort`, using `MessageEvent`, and then return.

4. Let messageClone be `deserializeRecord.[[Deserialized]]`.

5. Let newPorts be a new `frozen array` consisting of all `MessagePort` objects in `deserializeRecord.[[TransferredValues]]`, if any, maintaining their relative order.

6. Fire an event named `message` at `finalTargetPort`, using `MessageEvent`, with the `data` attribute initialized to `messageClone` and the `ports` attribute initialized to `newPorts`.

The `postMessage(message, options)` method, when invoked on a `MessagePort` object must run the following steps:

1. Let `targetPort` be the port with which this `MessagePort` is entangled, if any; otherwise let it be null.

2. Run the `message port post message steps` providing `targetPort`, `message` and `options`.

The `postMessage(message, transfer)` method, when invoked on a `MessagePort` object must run the following steps:

1. Let `targetPort` be the port with which this `MessagePort` is entangled, if any; otherwise let it be null.

2. Let `options` be «"transfer" → transfer ».

3. Run the `message port post message steps` providing `targetPort`, `message` and `options`.

The `start()` method, when invoked, must enable this `MessagePort` object's `port message queue`, if it is not already enabled.

The `close()` method, when invoked, must run these steps:

1. Set this `MessagePort` object's `[[Detached]]` internal slot value to true.

2. If this `MessagePort` object is entangled, disentangle it.

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 `MessagePort` interface:

<table>
<thead>
<tr>
<th>Event handler</th>
<th>Event handler event type</th>
</tr>
</thead>
<tbody>
<tr>
<td>onmessage</td>
<td><code>message</code></td>
</tr>
<tr>
<td>onmessageerror</td>
<td><code>messageerror</code></td>
</tr>
</tbody>
</table>

The first time a `MessagePort` object's `onmessage` IDL attribute is set, the port's `port message queue` must be enabled, as if the `start()` method had been called.

### 9.5.4 Broadcasting to many ports

This section is non-normative.

Broadcasting to many ports is in principle relatively simple: keep an array of `MessagePort` objects to send messages to, and iterate through the array to send a message. However, this has one rather unfortunate effect: it prevents the ports from being garbage collected, even if the other side has gone away. To avoid this problem, implement a simple protocol whereby the other side acknowledges it still exists. If it doesn't do so after a certain amount of time, assume it's gone, close the `MessagePort` object, and let it be garbage collected.
When a `MessagePort` object is entangled, user agents must either act as if o’s entangled `MessagePort` object has a strong reference to o, or as if o’s relevant global object has a strong reference to o.

Furthermore, a `MessagePort` object must not be garbage collected while there exists an event referenced by a `task` in a `task queue` that is to be dispatched on that `MessagePort` object, or while the `MessagePort` object's `port message queue` is enabled and not empty.

Authors are strongly encouraged to explicitly close `MessagePort` objects to disentangle them, so that their resources can be recollected. Creating many `MessagePort` objects and discarding them without closing them can lead to high transient memory usage since garbage collection is not necessarily performed promptly, especially for `MessagePort` s where garbage collection can involve cross-process coordination.

Pages on a single `origin` opened by the same user in the same user agent but in different unrelated `browsing contexts` sometimes need to send notifications to each other, for example “hey, the user logged in over here, check your credentials again”.

For elaborate cases, e.g. to manage locking of shared state, to manage synchronization of resources between a server and multiple local clients, to share a `WebSocket` connection with a remote host, and so forth, `shared workers` are the most appropriate solution.

For simple cases, though, where a shared worker would be an unreasonable overhead, authors can use the simple channel-based broadcast mechanism described in this section.

IDL

```idl
[Exposed=(Window,Worker)]
interface BroadcastChannel : EventTarget {
    constructor(DOMString name);

    readonly attribute DOMString name;
    undefined postMessage(any message);
    undefined close();
    attribute EventHandler onmessage;
    attribute EventHandler onmessageerror;
};
```

For web developers (non-normative)

```
broadcastChannel = new BroadcastChannel(name)

Returns a new `BroadcastChannel` object via which messages for the given channel name can be sent and received.

broadcastChannel . name

Returns the channel name (as passed to the constructor).

broadcastChannel . postMessage(message)

Sends the given message to other `BroadcastChannel` objects set up for this channel. Messages can be structured objects, e.g. nested objects and arrays.

broadcastChannel . close()

Closes the `BroadcastChannel` object, opening it up to garbage collection.
```
A `BroadcastChannel` object has a **channel name** and a **closed flag**.

The **new** `BroadcastChannel(name)` constructor steps are:

1. Set this's **channel name** to `name`.
2. Set this's **closed flag** to false.

The **name** getter steps are to return this's **channel name**.

The `postMessage(message)` method steps are:

1. If this's **closed flag** is true, then throw an "InvalidStateError" DOMException.
2. Let `serialized` be `StructuredSerialize(message)`. Rethrow any exceptions.
3. Let `sourceOrigin` be this's **relevant settings object's origin**.
4. Let `destinations` be a list of `BroadcastChannel` objects that match the following criteria:
   - Their **relevant global object** is either:
     - a `Window` object whose **associated Document** is **fully active**,
     - or a `WorkerGlobalScope` object whose **closing** flag is false and whose **worker** is not a suspendable worker.
   - Their **relevant settings object's origin** is **same origin** with `sourceOrigin`.
   - Their **channel name** is this's **channel name**.
5. Remove `source` from `destinations`.
6. Sort `destinations` such that all `BroadcastChannel` objects whose **relevant agents** are the same are sorted in creation order, oldest first. (This does not define a complete ordering. Within this constraint, user agents may sort the list in any implementation-defined manner.)
7. For each destination in `destinations`, queue a global task on the **DOM manipulation task source** given destination's **relevant global object** to perform the following steps:
   1. If destination's **closed flag** is true, then abort these steps.
   2. Let `targetRealm` be destination's **relevant Realm**.
   3. Let `data` be `StructuredDeserialize(serialized, targetRealm)`.
      If this throws an exception, catch it, fire an event named `messageerror` at destination, using `MessageEvent`, with the `origin` attribute initialized to the serialization of `sourceOrigin`, and then abort these steps.
   4. Fire an event named `message` at destination, using `MessageEvent`, with the `data` attribute initialized to `data` and the `origin` attribute initialized to the serialization of `sourceOrigin`.

While a `BroadcastChannel` object whose **closed flag** is false has an event listener registered for `message` or `messageerror` events, there must be a strong reference from the `BroadcastChannel` object's **relevant global object** to the `BroadcastChannel` object itself.

The **close()** method steps are to set this's **closed flag** to true.

---

Authors are strongly encouraged to explicitly close `BroadcastChannel` objects when they are no longer needed, so that they can be garbage collected. Creating many `BroadcastChannel` objects and discarding them while leaving them with an event listener and without closing them can lead to an apparent memory leak, since the objects will continue to live for as long as they have an event listener (or until their page or worker is closed).

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 `BroadcastChannel` interface:
Suppose a page wants to know when the user logs out, even when the user does so from another tab at the same site:

```javascript
var authChannel = new BroadcastChannel('auth');
authChannel.onmessage = function (event) {
  if (event.data == 'logout')
    showLogout();
}

function logoutRequested() {
  // called when the user asks us to log them out
  doLogout();
  showLogout();
  authChannel.postMessage('logout');
}

function doLogout() {
  // actually log the user out (e.g. clearing cookies)
  // ...
}

function showLogout() {
  // update the UI to indicate we're logged out
  // ...
}
```
10 Web workers

10.1 Introduction

10.1.1 Scope

This section is non-normative.

This specification defines an API for running scripts in the background independently of any user interface scripts.

This allows for long-running scripts that are not interrupted by scripts that respond to clicks or other user interactions, and allows long tasks to be executed without yielding to keep the page responsive.

Workers (as these background scripts are called herein) are relatively heavy-weight, and are not intended to be used in large numbers. For example, it would be inappropriate to launch one worker for each pixel of a four megapixel image. The examples below show some appropriate uses of workers.

Generally, workers are expected to be long-lived, have a high start-up performance cost, and a high per-instance memory cost.

10.1.2 Examples

This section is non-normative.

There are a variety of uses that workers can be put to. The following subsections show various examples of this use.

10.1.2.1 A background number-crunching worker

This section is non-normative.

The simplest use of workers is for performing a computationally expensive task without interrupting the user interface.

In this example, the main document spawns a worker to (naïvely) compute prime numbers, and progressively displays the most recently found prime number.

The main page is as follows:

```html
<!DOCTYPE HTML>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Worker example: One-core computation</title>
</head>
<body>
  <p>The highest prime number discovered so far is: <output id="result"></output></p>
  <script>
    var worker = new Worker('worker.js');
    worker.onmessage = function (event) {
      document.getElementById('result').textContent = event.data;
    }
  </script>
</body>
</html>
```

The `Worker()` constructor call creates a worker and returns a `Worker` object representing that worker, which is used to...
communicate with the worker. That object’s `onmessage` event handler allows the code to receive messages from the worker.

The worker itself is as follows:

```javascript
var n = 1;
search: while (true) {
    n += 1;
    for (var i = 2; i <= Math.sqrt(n); i += 1)
        if (n % i == 0)
            continue search;
    // found a prime!
    postMessage(n);
}
```

The bulk of this code is simply an unoptimized search for a prime number. The `postMessage()` method is used to send a message back to the page when a prime is found.

View this example online.

### 10.1.2.2 Using a JavaScript module as a worker

This section is non-normative.

All of our examples so far show workers that run classic scripts. Workers can instead be instantiated using module scripts, which have the usual benefits: the ability to use the JavaScript `import` statement to import other modules; strict mode by default; and top-level declarations not polluting the worker’s global scope.

As the `import` statement is available, the `importScripts()` method will automatically fail inside module workers.

In this example, the main document uses a worker to do off-main-thread image manipulation. It imports the filters used from another module.

The main page is as follows:

```html
<!DOCTYPE html>
<html lang="en">
<meta charset="utf-8">
<title>Worker example: image decoding</title>

<p>
    <label>
        Type an image URL to decode
        <input type="url" id="image-url" list="image-list">
        <datalist id="image-list">
            <option value="https://html.spec.whatwg.org/images/drawImage.png">
            <option value="https://html.spec.whatwg.org/images/robots.jpeg">
            <option value="https://html.spec.whatwg.org/images/arcTo2.png">
        </datalist>
    </label>
</p>

<p>
    <label>
        Choose a filter to apply
        <select id="filter">
            <option value="none">none</option>
            <option value="grayscale">grayscale</option>
            <option value="brighten">brighten by 20%</option>
        </select>
    </label>
</p>
```
The worker file is then:

```javascript
import * as filters from "./filters.js";

self.onmessage = e => {
    const { imageData, filter } = e.data;
    filters[filter](imageData);
    self.postMessage(imageData, [imageData.data.buffer]);
};

Which imports the file filters.js:

```javascript
export function none() {}
export function grayscale({ data: d }) {
    for (let i = 0; i < d.length; i += 4) {
```
```javascript
const [r, g, b] = [d[i], d[i + 1], d[i + 2]];

// CIE luminance for the RGB
// The human eye is bad at seeing red and blue, so we de-emphasize them.
d[i] = d[i + 1] = d[i + 2] = 0.2126 * r + 0.7152 * g + 0.0722 * b;
}

export function brighten({ data: d }) {
    for (let i = 0; i < d.length; ++i) {
        d[i] *= 1.2;
    }
};
```

View this example online.

10.1.2.3 Shared workers introduction

This section is non-normative.

This section introduces shared workers using a Hello World example. Shared workers use slightly different APIs, since each worker can have multiple connections.

This first example shows how you connect to a worker and how a worker can send a message back to the page when it connects to it. Received messages are displayed in a log.

Here is the HTML page:

```html
<!DOCTYPE HTML>
<html lang="en">
<meta charset="utf-8">
<title>Shared workers: demo 1</title>
<pre id="log">Log:</pre>
<script>
    var worker = new SharedWorker('test.js');
    var log = document.getElementById('log');
    worker.port.onmessage = function(e) {
        log.textContent += '\n' + e.data;
    }
</script>
</html>
```

Here is the JavaScript worker:

```javascript
onconnect = function(e) {
    var port = e.ports[0];
    port.postMessage('Hello World!');
}
```

View this example online.

This second example extends the first one by changing two things: first, messages are received using addEventListener() instead of an event handler IDL attribute, and second, a message is sent to the worker, causing the worker to send another message in return. Received messages are again displayed in a log.

Here is the HTML page:

```html
<!DOCTYPE HTML>
<html lang="en">
<meta charset="utf-8">
</html>
```
<title>Shared workers: demo 2</title>
<pre id="log">Log:</pre>
<script>
  var worker = new SharedWorker('test.js');
  var log = document.getElementById('log');
  worker.port.addEventListener('message', function(e) {
    log.textContent += '\n' + e.data;
  }, false);
  worker.port.start(); // note: need this when using addEventListener
  worker.port.postMessage('ping');
</script>

Here is the JavaScript worker:

onconnect = function(e) {
  var port = e.ports[0];
  port.postMessage('Hello World!');
  port.onmessage = function(e) {
    port.postMessage('pong'); // not e.ports[0].postMessage!
    // e.target.postMessage('pong'); would work also
  }
}

View this example online.

Finally, the example is extended to show how two pages can connect to the same worker; in this case, the second page is merely in an iframe on the first page, but the same principle would apply to an entirely separate page in a separate top-level browsing context.

Here is the outer HTML page:

<!DOCTYPE HTML>
<html lang="en">
<meta charset="utf-8">
<title>Shared workers: demo 3</title>
<pre id="log">Log:</pre>
<script>
  var worker = new SharedWorker('test.js');
  var log = document.getElementById('log');
  worker.port.addEventListener('message', function(e) {
    log.textContent += '\n' + e.data;
  }, false);
  worker.port.start(); // note: need this when using addEventListener
  worker.port.postMessage('ping');
</script>
<iframe src="inner.html"></iframe>

Here is the inner HTML page:

<!DOCTYPE HTML>
<html lang="en">
<meta charset="utf-8">
<title>Shared workers: demo 3 inner frame</title>
<pre id="log">Inner log:</pre>
<script>
  var worker = new SharedWorker('test.js');
  var log = document.getElementById('log');
  worker.port.onmessage = function(e) {
    log.textContent += '\n' + e.data;
  }
</script>
Here is the JavaScript worker:

```javascript
var count = 0;
onconnect = function(e) {
    count += 1;
    var port = e.ports[0];
    port.postMessage('Hello World! You are connection #' + count);
    port.onmessage = function(e) {
        port.postMessage('pong');
    }
}
```

View this example online.

10.1.2.4 Shared state using a shared worker

This section is non-normative.

In this example, multiple windows (viewers) can be opened that are all viewing the same map. All the windows share the same map information, with a single worker coordinating all the viewers. Each viewer can move around independently, but if they set any data on the map, all the viewers are updated.

The main page isn't interesting, it merely provides a way to open the viewers:

```html
<!DOCTYPE HTML>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Workers example: Multiviewer</title>
<script>
    function openViewer() {
        window.open('viewer.html');
    }
</script>
</head>
<body>
<p><button type=button onclick="openViewer()">Open a new viewer</button></p>
<p>Each viewer opens in a new window. You can have as many viewers as you like, they all view the same data.</p>
</body>
</html>

The viewer is more involved:

```html
<!DOCTYPE HTML>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Workers example: Multiviewer viewer</title>
<script>
    var worker = new SharedWorker('worker.js', 'core');

    // CONFIGURATION
    function configure(event) {
        if (event.data.substr(0, 4) != 'cfg ') return;
        var name = event.data.substr(4).split(' ', 1)[0];
```
// update display to mention our name is name
document.getElementsByTagName('h1')[0].textContent += ' ' + name;
// no longer need this listener
worker.port.removeEventListener('message', configure, false);
}
worker.port.addEventListener('message', configure, false);

// MAP
function paintMap(event) {
    if (event.data.substr(0, 4) !== 'map ') return;
    var data = event.data.substr(4).split(',');
    // display tiles data[0] .. data[8]
    var canvas = document.getElementById('map');
    var context = canvas.getContext('2d');
    for (var y = 0; y < 3; y += 1) {
        for (var x = 0; x < 3; x += 1) {
            var tile = data[y * 3 + x];
            if (tile == '0')
                context.fillStyle = 'green';
            else
                context.fillStyle = 'maroon';
            context.fillRect(x * 50, y * 50, 50, 50);
        }
    }
}
worker.port.addEventListener('message', paintMap, false);

// PUBLIC CHAT
function updatePublicChat(event) {
    if (event.data.substr(0, 4) !== 'txt ') return;
    var name = event.data.substr(4).split(' ', 1)[0];
    var message = event.data.substr(4 + name.length + 1);
    // display "<name> message" in public chat
    var public = document.getElementById('public');
    var p = document.createElement('p');
    var n = document.createElement('button');
    n.textContent = '<' + name + '>';
    n.onclick = function () { worker.port.postMessage('msg ' + name); };
    p.appendChild(n);
    var m = document.createElement('span');
    m.textContent = message;
    p.appendChild(m);
    public.appendChild(p);
}
worker.port.addEventListener('message', updatePublicChat, false);

// PRIVATE CHAT
function startPrivateChat(event) {
    if (event.data.substr(0, 4) !== 'msg ') return;
    var name = event.data.substr(4).split(' ', 1)[0];
    var port = event.ports[0];
    // display a private chat UI
    var ul = document.getElementById('private');
    var li = document.createElement('li');
    var h3 = document.createElement('h3');
    h3.textContent = 'Private chat with ' + name;
    li.appendChild(h3);
    var div = document.createElement('div');
    var addMessage = function(name, message) {
        var p = document.createElement('p');
        var n = document.createElement('strong');
        n.textContent = '<' + name + '>';
There are several key things worth noting about the way the viewer is written.

**Multiple listeners.** Instead of a single message processing function, the code here attaches multiple event listeners, each one
performing a quick check to see if it is relevant for the message. In this example it doesn't make much difference, but if multiple authors wanted to collaborate using a single port to communicate with a worker, it would allow for independent code instead of changes having to all be made to a single event handling function.

Registering event listeners in this way also allows you to unregister specific listeners when you are done with them, as is done with the \texttt{configure()} method in this example.

Finally, the worker:

```javascript
var nextName = 0;
function getNextName() {
    // this could use more friendly names
    // but for now just return a number
    return nextName++;
}

var map = [
    [0, 0, 0, 0, 0, 0, 0],
    [1, 1, 0, 1, 0, 1, 1],
    [0, 1, 1, 0, 0, 0, 0],
    [0, 1, 0, 1, 0, 1, 1],
    [0, 0, 0, 1, 0, 0, 0],
    [1, 0, 0, 1, 1, 1, 1],
    [1, 1, 0, 1, 1, 0, 1],
];

function wrapX(x) {
    if (x < 0) return wrapX(x + map[0].length);
    if (x >= map[0].length) return wrapX(x - map[0].length);
    return x;
}

function wrapY(y) {
    if (y < 0) return wrapY(y + map.length);
    if (y >= map[0].length) return wrapY(y - map.length);
    return y;
}

function wrap(val, min, max) {
    if (val < min)
        return val + (max-min)+1;
    if (val > max)
        return val - (max-min)-1;
    return val;
}

function sendMapData(viewer) {
    var data = '';
    for (var y = viewer.y-1; y <= viewer.y+1; y += 1) {
        for (var x = viewer.x-1; x <= viewer.x+1; x += 1) {
            if (data != '')
                data += ',';
            data += map[wrap(y, 0, map[0].length-1)][wrap(x, 0, map.length-1)];
        }
    }
    viewer.port.postMessage('map ' + data);
}

var viewers = {};
onconnect = function (event) {
    var name = getNextName();
    event.ports[0]._data = { port: event.ports[0], name: name, x: 0, y: 0, };
    viewers[name] = event.ports[0]._data;
```
```javascript
function getMessage(event) {
  switch (event.data.substr(0, 4)) {
    case 'mov ':  
      var direction = event.data.substr(4);  
      var dx = 0;  
      var dy = 0;  
      switch (direction) {  
        case 'up' : dy = -1; break;  
        case 'down': dy = 1; break;  
        case 'left' : dx = -1; break;  
        case 'right': dx = 1; break;  
      }  
      event.target._data.x = wrapX(event.target._data.x + dx);  
      event.target._data.y = wrapY(event.target._data.y + dy);  
      sendMapData(event.target._data);  
      break;  
    case 'set ':  
      var value = event.data.substr(4);  
      map[event.target._data.y][event.target._data.x] = value;  
      for (var viewer in viewers)  
        sendMapData(viewers[viewer]);  
      break;  
    case 'txt ':  
      var name = event.target._data.name;  
      var message = event.data.substr(4);  
      for (var viewer in viewers)  
        viewers[viewer].port.postMessage('txt ' + name + ' ' + message);  
      break;  
    case 'msg ':  
      var party1 = event.target._data;  
      var party2 = viewers[event.data.substr(4).split(', 1')[0]];  
      if (party2) {  
        var channel = new MessageChannel();  
        party1.port.postMessage('msg ' + party2.name, [channel.port1]);  
        party2.port.postMessage('msg ' + party1.name, [channel.port2]);  
      }  
      break;  
  }
}
```

Connecting to multiple pages. The script uses the `onconnect` event listener to listen for multiple connections.

Direct channels. When the worker receives a "msg" message from one viewer naming another viewer, it sets up a direct connection between the two, so that the two viewers can communicate directly without the worker having to proxy all the messages.

View this example online.

10.1.2.5 Delegation

This section is non-normative.

With multicore CPUs becoming prevalent, one way to obtain better performance is to split computationally expensive tasks amongst multiple workers. In this example, a computationally expensive task that is to be performed for every number from 1 to 10,000,000 is farmed out to ten subworkers.

The main page is as follows, it just reports the result:
The worker itself is as follows:

```
// settings
var num_workers = 10;
var items_per_worker = 1000000;

// start the workers
var result = 0;
var pending_workers = num_workers;
for (var i = 0; i < num_workers; i += 1) {
  var worker = new Worker('core.js');
  worker.postMessage(i * items_per_worker);
  worker.postMessage((i+1) * items_per_worker);
  worker.onmessage = storeResult;
}

// handle the results
function storeResult(event) {
  result += 1*event.data;
  pending_workers -= 1;
  if (pending_workers <= 0)
    postMessage(result); // finished!
}
```

It consists of a loop to start the subworkers, and then a handler that waits for all the subworkers to respond.

The subworkers are implemented as follows:

```
var start;
onmessage = getStart;
function getStart(event) {
  start = 1*event.data;
  onmessage = getEnd;
}

var end;
function getEnd(event) {
  end = 1*event.data;
  onmessage = null;
  work();
}

function work() {
  var result = 0;
  for (var i = start; i < end; i += 1) {
```
They receive two numbers in two events, perform the computation for the range of numbers thus specified, and then report the result back to the parent.

View this example online.

10.1.2.6 Providing libraries

This section is non-normative.

Suppose that a cryptography library is made available that provides three tasks:

**Generate a public/private key pair**

Takes a port, on which it will send two messages, first the public key and then the private key.

**Given a plaintext and a public key, return the corresponding ciphertext**

Takes a port, to which any number of messages can be sent, the first giving the public key, and the remainder giving the plaintext, each of which is encrypted and then sent on that same channel as the ciphertext. The user can close the port when it is done encrypting content.

**Given a ciphertext and a private key, return the corresponding plaintext**

Takes a port, to which any number of messages can be sent, the first giving the private key, and the remainder giving the ciphertext, each of which is decrypted and then sent on that same channel as the plaintext. The user can close the port when it is done decrypting content.

The library itself is as follows:

```javascript
// perform some complex calculation here
result += 1;
}
postMessage(result);
close();
}

They receive two numbers in two events, perform the computation for the range of numbers thus specified, and then report the result back to the parent.

View this example online.

10.1.2.6 Providing libraries

This section is non-normative.

Suppose that a cryptography library is made available that provides three tasks:

**Generate a public/private key pair**

Takes a port, on which it will send two messages, first the public key and then the private key.

**Given a plaintext and a public key, return the corresponding ciphertext**

Takes a port, to which any number of messages can be sent, the first giving the public key, and the remainder giving the plaintext, each of which is encrypted and then sent on that same channel as the ciphertext. The user can close the port when it is done encrypting content.

**Given a ciphertext and a private key, return the corresponding plaintext**

Takes a port, to which any number of messages can be sent, the first giving the private key, and the remainder giving the ciphertext, each of which is decrypted and then sent on that same channel as the plaintext. The user can close the port when it is done decrypting content.

The library itself is as follows:

```javascript
function handleMessage(e) {
    if (e.data == "genkeys")
        genkeys(e.ports[0]);
    else if (e.data == "encrypt")
        encrypt(e.ports[0]);
    else if (e.data == "decrypt")
        decrypt(e.ports[0]);
}

function genkeys(p) {
    var keys = _generateKeyPair();
    p.postMessage(keys[0]);
    p.postMessage(keys[1]);
}

function encrypt(p) {
    var key, state = 0;
    p.onmessage = function (e) {
        if (state == 0) {
            key = e.data;
            state = 1;
        } else {
            p.postMessage(_encrypt(key, e.data));
        }
    }
}

function decrypt(p) {
```
var key, state = 0;
p.onmessage = function (e) {
  if (state == 0) {
    key = e.data;
    state = 1;
  } else {
    p.postMessage(_decrypt(key, e.data));
  }
};

// support being used as a shared worker as well as a dedicated worker
if ('onmessage' in this) // dedicated worker
  onmessage = handleMessage;
else // shared worker
  onconnect = function (e) { e.port.onmessage = handleMessage; }

// the "crypto" functions:

function _generateKeyPair() {
  return [Math.random(), Math.random()];
}

function _encrypt(k, s) {
  return 'encrypted-' + k + ' ' + s;
}

function _decrypt(k, s) {
  return s.substr(s.indexOf(' ') + 1);
}

Note that the crypto functions here are just stubs and don't do real cryptography.

This library could be used as follows:

<!DOCTYPE HTML>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Worker example: Crypto library</title>
  <script>
    const cryptoLib = new Worker('libcrypto-v1.js'); // or could use 'libcrypto-v2.js'
    function startConversation(source, message) {
      const messageChannel = new MessageChannel();
      source.postMessage(message, [messageChannel.port2]);
      return messageChannel.port1;
    }
    function getKeys() {
      let state = 0;
      startConversation(cryptoLib, "genkeys").onmessage = function (e) {
        if (state == 0)
          document.getElementById('public').value = e.data;
        else if (state == 1)
          document.getElementById('private').value = e.data;
        state += 1;
      };
    }
    function enc() {
      const port = startConversation(cryptoLib, "encrypt");
      port.postMessage(document.getElementById('public').value);
      port.postMessage(document.getElementById('input').value);
    }
  </script>
</head>
<body>
</body>
</html>
A later version of the API, though, might want to offload all the crypto work onto subworkers. This could be done as follows:

```javascript
function handleMessage(e) {
  if (e.data == "genkeys")
    genkeys(e.ports[0]);
  else if (e.data == "encrypt")
    encrypt(e.ports[0]);
  else if (e.data == "decrypt")
    decrypt(e.ports[0]);
}

function genkeys(p) {
  var generator = new Worker('libcrypto-v2-generator.js');
  generator.postMessage('', [p]);
}

function encrypt(p) {
  p.onmessage = function (e) {
    var key = e.data;
    var encryptor = new Worker('libcrypto-v2-encryptor.js');
    encryptor.postMessage(key, [p]);
  }
}

function decrypt(p) {
  p.onmessage = function (e) {
    var key = e.data;
    var decryptor = new Worker('libcrypto-v2-decryptor.js');
    decryptor.postMessage(key, [p]);
  }
}
```
// support being used as a shared worker as well as a dedicated worker
if ('onmessage' in this) // dedicated worker
  onmessage = handleMessage;
else // shared worker
  onconnect = function (e) { e.ports[0].onmessage = handleMessage;};

The little subworkers would then be as follows.

For generating key pairs:

```javascript
onmessage = function (e) {
  var k = _generateKeyPair();
  e.ports[0].postMessage(k[0]);
  e.ports[0].postMessage(k[1]);
  close();
}

generateKeyPair() {
  return [Math.random(), Math.random()];
}
```

For encrypting:

```javascript
onmessage = function (e) {
  var key = e.data;
  e.ports[0].onmessage = function (e) {
    var s = e.data;
    postMessage(_encrypt(key, s));
  }
}

_encrypt(k, s) {
  return 'encrypted-' + k + ' ' + s;
}
```

For decrypting:

```javascript
onmessage = function (e) {
  var key = e.data;
  e.ports[0].onmessage = function (e) {
    var s = e.data;
    postMessage(_decrypt(key, s));
  }
}

_decrypt(k, s) {
  return s.substr(s.indexOf('') + 1);
}
```

Notice how the users of the API don’t have to even know that this is happening — the API hasn’t changed; the library can delegate to subworkers without changing its API, even though it is accepting data using message channels.

View this example online.

10.1.3 Tutorials

10.1.3.1 Creating a dedicated worker

This section is non-normative.
Creating a worker requires a URL to a JavaScript file. The `Worker()` constructor is invoked with the URL to that file as its only argument; a worker is then created and returned:

```javascript
var worker = new Worker('helper.js');
```

If you want your worker script to be interpreted as a module script instead of the default classic script, you need to use a slightly different signature:

```javascript
var worker = new Worker('helper.mjs', { type: "module" });
```

### 10.1.3.2 Communicating with a dedicated worker

This section is non-normative.

Dedicated workers use `MessagePort` objects behind the scenes, and thus support all the same features, such as sending structured data, transferring binary data, and transferring other ports.

To receive messages from a dedicated worker, use the `onmessage` event handler IDL attribute on the `Worker` object:

```javascript
worker.onmessage = function (event) { ... };
```

You can also use the `addEventListener()` method.

**Note**

The implicit `MessagePort` used by dedicated workers has its `port message queue` implicitly enabled when it is created, so there is no equivalent to the `MessagePort` interface's `start()` method on the `Worker` interface.

To send data to a worker, use the `postMessage()` method. Structured data can be sent over this communication channel. To send `ArrayBuffer` objects efficiently (by transferring them rather than cloning them), list them in an array in the second argument.

```javascript
worker.postMessage({
  operation: 'find-edges',
  input: buffer, // an ArrayBuffer object
  threshold: 0.6,
}, [buffer]);
```

To receive a message inside the worker, the `onmessage` event handler IDL attribute is used.

```javascript
onmessage = function (event) { ... };
```

You can again also use the `addEventListener()` method.

In either case, the data is provided in the event object's `data` attribute.

To send messages back, you again use `postMessage()`. It supports the structured data in the same manner.

```javascript
postMessage(event.data.input, [event.data.input]); // transfer the buffer back
```

### 10.1.3.3 Shared workers

This section is non-normative.

Shared workers are identified by the URL of the script used to create it, optionally with an explicit name. The name allows multiple instances of a particular shared worker to be started.

Shared workers are scoped by `origin`. Two different sites using the same names will not collide. However, if a page tries to use the
same shared worker name as another page on the same site, but with a different script URL, it will fail.

Creating shared workers is done using the `SharedWorker()` constructor. This constructor takes the URL to the script to use for its first argument, and the name of the worker, if any, as the second argument.

```javascript
var worker = new SharedWorker('service.js');
```

Communicating with shared workers is done with explicit `MessagePort` objects. The object returned by the `SharedWorker()` constructor holds a reference to the port on its `port` attribute.

```javascript
worker.port.onmessage = function(event) {
    ...
};
worker.port.postMessage('some message');
worker.port.postMessage({ foo: 'structured', bar: ['data', 'also', 'possible'] });
```

Inside the shared worker, new clients of the worker are announced using the `connect` event. The port for the new client is given by the event object's `source` attribute.

```javascript
onconnect = function(event) {
    var newPort = event.source;
    // set up a listener
    newPort.onmessage = function(event) {
        ...
    }; // send a message back to the port
    newPort.postMessage('ready!'); // can also send structured data, of course
};
```

10.2 Infrastructure

This standard defines two kinds of workers: dedicated workers, and shared workers. Dedicated workers, once created, are linked to their creator, but message ports can be used to communicate from a dedicated worker to multiple other browsing contexts or workers. Shared workers, on the other hand, are named, and once created any script running in the same `origin` can obtain a reference to that worker and communicate with it. `Service Workers` defines a third kind. [SWI]

10.2.1 The global scope

The global scope is the "inside" of a worker.

10.2.1.1 The `WorkerGlobalScope` common interface

```idl
[Exposed=Worker]
interface WorkerGlobalScope : EventTarget {
    readonly attribute WorkerGlobalScope self;
    readonly attribute WorkerLocation location;
    readonly attribute Navigator navigator;
    undefined importScripts(USVString... urls);
    attribute OnErrorEventHandler onerror;
    attribute EventHandler onlanguagechange;
    attribute EventHandler onoffline;
    attribute EventHandler ononline;
    attribute EventHandler onrejectionhandled;
    attribute EventHandler onunhandledrejection;
};
```

`WorkerGlobalScope` serves as the base class for specific types of worker global scope objects, including...
A **WorkerGlobalScope** object has an associated **owner set** (a set of **Document** and **WorkerGlobalScope** objects). It is initially empty and populated when the worker is created or obtained.

**Note**

It is a set, instead of a single owner, to accommodate **SharedWorkerGlobalScope** objects.

A **WorkerGlobalScope** object has an associated **worker set** (a set of **WorkerGlobalScope** objects). It is initially empty and populated when the worker creates or obtains further workers.

A **WorkerGlobalScope** object has an associated **type** ("classic" or "module"). It is set during creation.

A **WorkerGlobalScope** object has an associated **url** (null or a **URL**). It is initially null.

A **WorkerGlobalScope** object has an associated **name** (a string). It is set during creation.

**Note**

The **name** can have different semantics for each subclass of **WorkerGlobalScope**. For **DedicatedWorkerGlobalScope** instances, it is simply a developer-supplied name, useful mostly for debugging purposes. For **SharedWorkerGlobalScope** instances, it allows obtaining a reference to a common shared worker via the **SharedWorker()** constructor. For **ServiceWorkerGlobalScope** objects, it doesn't make sense (and as such isn't exposed through the JavaScript API at all).

A **WorkerGlobalScope** object has an associated **referrer policy** (a **referrer policy**). It is initially the empty string.

A **WorkerGlobalScope** object has an associated **embedder policy** (an **embedder policy**).

A **WorkerGlobalScope** object has an associated **CSP list**, which is a **CSP list** containing all of the **Content Security Policy** objects active for the worker. It is initially an empty list.

A **WorkerGlobalScope** object has an associated **module map**. It is a **module map**, initially empty.

A **WorkerGlobalScope** object has an associated **cross-origin isolated capability** boolean. It is initially false.

**For web developers (non-normative)**

**workerGlobal**. **self**

Returns **workerGlobal**.

**workerGlobal**. **location**

Returns **workerGlobal**'s **WorkerLocation** object.

**workerGlobal**. **navigator**

Returns **workerGlobal**'s **WorkerNavigator** object.

**workerGlobal**. **importScripts**(urls...)

Fetches each **URL** in **urls**, executes them one-by-one in the order they are passed, and then returns (or throws if something went amiss).

The **self** attribute must return the **WorkerGlobalScope** object itself.

The **location** attribute must return the **WorkerLocation** object whose associated **WorkerGlobalScope object** is the **WorkerGlobalScope** object.

**Note**

While the **WorkerLocation** object is created after the **WorkerGlobalScope** object, this is not problematic as it cannot be observed from script.

The following are the **event handlers** (and their corresponding **event handler event types**) that must be supported, as **event handler IDL attributes**, by objects implementing the **WorkerGlobalScope** interface:
<table>
<thead>
<tr>
<th>Event handler</th>
<th>Event handler event type</th>
</tr>
</thead>
<tbody>
<tr>
<td>onerror</td>
<td>error</td>
</tr>
<tr>
<td>onlanguagechange</td>
<td>languagechange</td>
</tr>
<tr>
<td>onoffline</td>
<td>offline</td>
</tr>
<tr>
<td>ononline</td>
<td>online</td>
</tr>
<tr>
<td>onrejectionhandled</td>
<td>rejectionhandled</td>
</tr>
<tr>
<td>onunhandledrejection</td>
<td>unhandledrejection</td>
</tr>
</tbody>
</table>

### 10.2.1.2 Dedicated workers and the DedicatedWorkerGlobalScope interface

**IDL**

```idl
interface DedicatedWorkerGlobalScope : WorkerGlobalScope {
  [Replaceable] readonly attribute DOMString name;

  postMessage(any message, sequence<object> transfer);
  postMessage(any message, optional PostMessageOptions options = {});

  close();

  attribute EventHandler onmessage;
  attribute EventHandler onmessageerror;
};
```

DedicatedWorkerGlobalScope objects act as if they had an implicit MessagePort associated with them. This port is part of a channel that is set up when the worker is created, but it is not exposed. This object must never be garbage collected before the DedicatedWorkerGlobalScope object.

All messages received by that port must immediately be retargeted at the DedicatedWorkerGlobalScope object.

**For web developers (non-normative)**

```javascript
dedicatedWorkerGlobal . name

 Returns dedicatedWorkerGlobal's name, i.e. the value given to the Worker constructor. Primarily useful for debugging.

dedicatedWorkerGlobal . postMessage(message [, transfer ])
dedicatedWorkerGlobal . postMessage(message [, { transfer }])

 Clones message and transmits it to the Worker object associated with dedicatedWorkerGlobal. transfer can be passed as a list of objects that are to be transferred rather than cloned.

dedicatedWorkerGlobal . close()

 Aborts dedicatedWorkerGlobal.
```

The name attribute must return the DedicatedWorkerGlobalScope object's name. Its value represents the name given to the worker using the Worker constructor, used primarily for debugging purposes.

The postMessage(message, transfer) and postMessage(message, options) methods on DedicatedWorkerGlobalScope objects act as if, when invoked, it immediately invoked the respective postMessage(message, transfer) and postMessage(message, options) on the port, with the same arguments, and returned the same return value.

To close a worker, given a workerGlobal, run these steps:

1. Discard any tasks that have been added to workerGlobal's relevant agent's event loop's task queues.
2. Set workerGlobal's closing flag to true. (This prevents any further tasks from being queued.)

The close() method, when invoked, must close a worker with this DedicatedWorkerGlobalScope object.

The following are the event handlers (and their corresponding event handler event types) that must be supported, as event handler IDL attributes, by objects implementing the DedicatedWorkerGlobalScope interface:
### 10.2.1.3 Shared workers and the SharedWorkerGlobalScope Interface

A `SharedWorkerGlobalScope` object has an associated `constructor origin`, `constructor url`, and `credentials`. They are initialized when the `SharedWorkerGlobalScope` object is created, in the `run a worker` algorithm.

Shared workers receive message ports through `connect` events on their `SharedWorkerGlobalScope` object for each connection.

#### For web developers (non-normative)

**sharedWorkerGlobal . name**

Returns `sharedWorkerGlobal`'s `name`, i.e. the value given to the `SharedWorker` constructor. Multiple `SharedWorker` objects can correspond to the same shared worker (and `SharedWorkerGlobalScope`), by reusing the same name.

**sharedWorkerGlobal . close**

Aborts `sharedWorkerGlobal`.

The `name` attribute must return the `SharedWorkerGlobalScope` object's `name`, i.e. the value that can be used to obtain a reference to the worker using the `SharedWorker` constructor.

The `close()` method, when invoked, must close a worker with this `SharedWorkerGlobalScope` object.

The following are the `event handlers` (and their corresponding `event handler event types`) that must be supported, as `event handler IDL attributes`, by objects implementing the `SharedWorkerGlobalScope` interface:

<table>
<thead>
<tr>
<th>Event handler</th>
<th>Event handler event type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>onconnect</code></td>
<td><code>connect</code></td>
</tr>
</tbody>
</table>

### 10.2.2 The event loop

A `worker event loop`'s `task queues`, only have events, callbacks, and networking activity as `tasks`. These `worker event loops` are created by the `run a worker` algorithm.

Each `WorkerGlobalScope` object has a `closing` flag, which must be initially false, but which can get set to true by the algorithms in the processing model section below.

Once the `WorkerGlobalScope`'s `closing` flag is set to true, the `event loop`'s `task queues` must discard any further `tasks` that would be added to them (tasks already on the queue are unaffected except where otherwise specified). Effectively, once the `closing` flag is true, timers stop firing, notifications for all pending background operations are dropped, etc.
10.2.3 The worker's lifetime

Workers communicate with other workers and with browsing contexts through message channels and their MessagePort objects.

Each WorkerGlobalScope object worker global scope has a list of the worker's ports, which consists of all the MessagePort objects that are entangled with another port and that have one (but only one) port owned by worker global scope. This list includes the implicit MessagePort in the case of dedicated workers.

Given an environment settings object o when creating or obtaining a worker, the relevant owner to add depends on the type of global object specified by o. If o specifies a global object that is a WorkerGlobalScope object (i.e., if we are creating a nested dedicated worker), then the relevant owner is that global object. Otherwise, o specifies a global object that is a Window object, and the relevant owner is the responsible document specified by o.

A worker is said to be a permissible worker if its WorkerGlobalScope's owner set is not empty or:

- its owner set has been empty for no more than a short implementation-defined timeout value,
- its WorkerGlobalScope object is a SharedWorkerGlobalScope object (i.e., the worker is a shared worker), and
- the user agent has a browsing context whose Document object is not completely loaded.

A worker is said to be an active needed worker if any its owners are either Document objects that are fully active or active needed workers.

A worker is said to be a protected worker if it is an active needed worker and either it has outstanding timers, database transactions, or network connections, or its list of the worker's ports is not empty, or its WorkerGlobalScope is actually a SharedWorkerGlobalScope object (i.e., the worker is a shared worker).

A worker is said to be a suspendable worker if it is not an active needed worker but it is a permissible worker.

10.2.4 Processing model

When a user agent is to run a worker for a script with Worker or SharedWorker object worker, URL url, environment settings object outside settings, MessagePort outside port, and a WorkerOptions dictionary options, it must run the following steps.

1. Let is shared be true if worker is a SharedWorker object, and false otherwise.
2. Let owner be the relevant owner to add given outside settings.
3. Let parent worker global scope be null.
4. If owner is a WorkerGlobalScope object (i.e., we are creating a nested dedicated worker), then set parent worker global scope to owner.
5. Let agent be the result of obtaining a dedicated/shared worker agent given outside settings and is shared. Run the rest of these steps in that agent.
   For the purposes of timing APIs, this is the official moment of creation of the worker.
6. Let realm execution context be the result of creating a new JavaScript realm given agent and the following customizations:
   - For the global object, if is shared is true, create a new SharedWorkerGlobalScope object. Otherwise, create a new DedicatedWorkerGlobalScope object.
7. Let worker global scope be the global object of realm execution context's Realm component.
8. Set up a worker environment settings object with realm execution context and outside settings, and let inside settings be the result.

9. Set worker global scope’s name to the value of options’s name member.

10. Append owner to worker global scope’s owner set.

11. If parent worker global scope is not null, then append worker global scope to parent worker global scope’s worker set.

12. If is shared is true, then:
   1. Set worker global scope’s constructor origin to outside settings’s origin.
   2. Set worker global scope’s constructor url to url.
   3. Set worker global scope’s type to the value of options’s type member.
   4. Set worker global scope’s credentials to the value of options’s credentials member.

13. Let destination be "sharedworker" if is shared is true, and "worker" otherwise.

14. Obtain script by switching on the value of options’s type member:
   - "classic"  
     Fetch a classic worker script given url, outside settings, destination, and inside settings.
   - "module"  
     Fetch a module worker script graph given url, outside settings, destination, the value of the credentials member of options, and inside settings.

In both cases, to perform the fetch given request, perform the following steps if the is top-level flag is set:

1. Set request’s reserved client to inside settings.

2. Fetch request, and asynchronously wait to run the remaining steps as part of fetch’s process response for the response.

3. Set worker global scope’s url to response’s url.

4. Set worker global scope’s referrer policy to the result of parsing the `Referrer-Policy` header of response.

5. If response’s url’s scheme is a local scheme, then set worker global scope’s embedder policy to owner’s embedder policy.

6. Otherwise, set worker global scope’s embedder policy to the result of obtaining an embedder policy from response and inside settings.

7. If worker global scope’s embedder policy is "require-corp" and is shared is true, then set agent’s agent cluster’s cross-origin isolation mode to "logical" or "concrete". The one chosen is implementation-defined.

8. If the result of checking a global object’s embedder policy with worker global scope, owner, and response is false, then set response to a network error.

9. Set worker global scope’s cross-origin isolated capability to true if agent’s agent cluster’s cross-origin isolation mode is "concrete".

10. If is shared is false and owner’s cross-origin isolated capability is false, then set worker global scope’s cross-origin isolated capability to false.

11. If is shared is false and response’s url’s scheme is "data", then set worker global scope’s cross-origin isolated capability to false.
12. **Initialize a global object's CSP list** given worker global scope and response. 

13. Asynchronously complete the **perform the fetch** steps with response.

If the algorithm asynchronously completes with null or with a script whose **error to rethrow** is non-null, then:

1. **Queue a global task** on the DOM manipulation task source given worker's relevant global object to fire an event named **error** at worker.
2. Run the **environment discarding steps** for inside settings.
3. Return.

Otherwise, continue the rest of these steps after the algorithm's asynchronous completion, with script being the asynchronous completion value.

15. Associate worker with worker global scope.

16. Let inside port be a new MessagePort object in inside settings's Realm.

17. Associate inside port with worker global scope.

18. **Entangle** outside port and inside port.

19. Create a new WorkerLocation object and associate it with worker global scope.

20. **Closing orphan workers**: Start monitoring the worker such that no sooner than it stops being a protected worker, and no later than it stops being a permissible worker, worker global scope's closing flag is set to true.

21. **Suspending workers**: Start monitoring the worker, such that whenever worker global scope's closing flag is false and the worker is a suspendable worker, the user agent suspends execution of script in that worker until such time as either the closing flag switches to true or the worker stops being a suspendable worker.

22. Set inside settings's execution ready flag.

23. If script is a classic script, then run the classic script script. Otherwise, it is a module script; run the module script script.

24. Enable outside port's port message queue.

25. If is shared is false, enable the port message queue of the worker's implicit port.

26. If is shared is true, then queue a global task on DOM manipulation task source given worker global scope to fire an event named connect at worker global scope, using MessageEvent, with the data attribute initialized to the empty string, the ports attribute initialized to a new frozen array containing inside port, and the source attribute initialized to inside port.

27. Enable the client message queue of the ServiceWorkerContainer object whose associated service worker client is worker global scope's relevant settings object.

28. **Event loop**: Run the responsible event loop specified by inside settings until it is destroyed.

**Note**

In addition to the usual possibilities of returning a value or failing due to an exception, this could be prematurely aborted by the terminate a worker algorithm defined below.

**Note**

The handling of events or the execution of callbacks by tasks, run by the event loop, might get prematurely aborted by the terminate a worker algorithm defined below.

**Note**

The worker processing model remains on this step until the event loop is destroyed, which happens after the closing flag is set to true.
29. Empty the worker global scope’s list of active timers.
30. Disentangle all the ports in the list of the worker’s ports.
31. Empty worker global scope’s owner set.

When a user agent is to terminate a worker it must run the following steps in parallel with the worker’s main loop (the “run a worker” processing model defined above):

1. Set the worker’s WorkerGlobalScope object’s closing flag to true.
2. If there are any tasks queued in the WorkerGlobalScope object’s relevant agent’s event loop’s task queues, discard them without processing them.
3. Abort the script currently running in the worker.
4. If the worker’s WorkerGlobalScope object is actually a DedicatedWorkerGlobalScope object (i.e. the worker is a dedicated worker), then empty the port message queue of the port that the worker’s implicit port is entangled with.

User agents may invoke the terminate a worker algorithm when a worker stops being an active needed worker and the worker continues executing even after its closing flag was set to true.

10.2.5 Runtime script errors

Whenever an uncaught runtime script error occurs in one of the worker’s scripts, if the error did not occur while handling a previous script error, the user agent must report the error for that script, with the position (line number and column number) where the error occurred, using the WorkerGlobalScope object as the target.

For shared workers, if the error is still not handled afterwards, the error may be reported to a developer console.

For dedicated workers, if the error is still not handled afterwards, the user agent must queue a task to run these steps:

1. Let notHandled be the result of firing an event named error at the Worker object associated with the worker, using ErrorEvent, with the cancelable attribute initialized to true, the message, filename, lineno, and colno attributes initialized appropriately, and the error attribute initialized to null.
2. If notHandled is true, then the user agent must act as if the uncaught runtime script error had occurred in the global scope that the Worker object is in, thus repeating the entire runtime script error reporting process one level up.

If the implicit port connecting the worker to its Worker object has been disentangled (i.e. if the parent worker has been terminated), then the user agent must act as if the Worker object had no error event handler and as if that worker’s onerror attribute was null, but must otherwise act as described above.

Note

Thus, error reports propagate up to the chain of dedicated workers up to the original Document, even if some of the workers along this chain have been terminated and garbage collected.

The task source for the task mentioned above is the DOM manipulation task source.

10.2.6 Creating workers

10.2.6.1 The AbstractWorker mixin

IDL

interface mixin AbstractWorker {
    attribute EventHandler onerror;
}
The following are the event handlers\(^{p941}\) (and their corresponding event handler event types\(^{p943}\)) that must be supported, as event handler IDL attributes\(^{p942}\), by objects implementing the AbstractWorker\(^{p1034}\) interface:

<table>
<thead>
<tr>
<th>Event handler(^{p941})</th>
<th>Event handler event type(^{p943})</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>onerror</code></td>
<td><code>error</code></td>
</tr>
</tbody>
</table>

### 10.2.6.2 Script settings for workers\(^{p1035}\)

To set up a worker environment settings object, given a JavaScript execution context execution context and environment settings object\(^{p903}\) outside settings:

1. Let `inherited origin` be `outside settings`'s `origin`\(^{p903}\).
2. Let `realm` be the value of execution context's Realm component.
3. Let `worker global scope` be `realm`'s `global object`\(^{p904}\).
4. Let `settings object` be a new environment settings object\(^{p903}\) whose algorithms are defined as follows:
   - **The realm execution context**\(^{p903}\)
     Return execution context.
   - **The module map**\(^{p903}\)
     Return worker global scope's `module map`\(^{p1028}\).
   - **The responsible document**\(^{p903}\)
     Not applicable (the responsible event loop\(^{p904}\) is not a window event loop\(^{p931}\)).
   - **The API URL character encoding**\(^{p903}\)
     Return UTF-8.
   - **The API base URL**\(^{p903}\)
     Return worker global scope's `url`\(^{p1028}\).
   - **The origin**\(^{p903}\)
     Return a unique opaque origin\(^{p840}\) if worker global scope's `url`\(^{p1028}\)'s `scheme` is "data", and `inherited origin` otherwise.
   - **The referrer policy**\(^{p903}\)
     Return worker global scope's `referrer policy`\(^{p1028}\).
   - **The embedder policy**\(^{p903}\)
     Return worker global scope's `embedder policy`\(^{p1028}\).
   - **The cross-origin isolated capability**\(^{p903}\)
     Return worker global scope's `cross-origin isolated capability`\(^{p1028}\).

5. Set `settings object`'s `id`\(^{p902}\) to a new unique opaque string, `creation URL`\(^{p902}\) to `worker global scope`'s `url`, `top-level creation URL`\(^{p902}\) to null, `target browsing context`\(^{p903}\) to null, and `active service worker`\(^{p903}\) to null.
6. If `worker global scope` is a DedicatedWorkerGlobalScope\(^{p1828}\) object, then set `settings object`'s `top-level origin`\(^{p902}\) to `outside settings`'s `top-level origin`\(^{p902}\).
7. Otherwise, set `settings object`'s `top-level origin`\(^{p902}\) to an implementation-defined value.

See Client-Side Storage Partitioning for the latest on properly defining this.

8. Set `realm`'s `[[HostDefined]]` field to `settings object`.
9. Return `settings object`.
### IDL

```idl
[Exposed=(Window, DedicatedWorker, SharedWorker)]

interface Worker : EventTarget {
    constructor(USVString scriptURL, optional WorkerOptions options = {});

    undefined terminate();

    undefined postMessage(any message, sequence<object> transfer);
    undefined postMessage(any message, optional PostMessageOptions options = {});
    attribute EventHandler onmessage;
    attribute EventHandler onmessageerror;
};

dictionary WorkerOptions {
    WorkerType type = "classic";
    RequestCredentials credentials = "same-origin"; // credentials is only used if type is "module"
    DOMString name = "";
};

eenum WorkerType { "classic", "module" }

Worker includes AbstractWorker;
```

### For web developers (non-normative)

```javascript
worker = new Worker(scriptURL [, options ])
```

Returns a new `Worker` object. `scriptURL` will be fetched and executed in the background, creating a new global environment for which `worker` represents the communication channel. `options` can be used to define the `name` of that global environment via the name option, primarily for debugging purposes. It can also ensure this new global environment supports JavaScript modules (specify type: "module"), and if that is specified, can also be used to specify how `scriptURL` is fetched through the credentials option.

```javascript
worker.terminate()
```

Aborts `worker`'s associated global environment.

```javascript
worker.postMessage(message [, transfer ])
worker.postMessage(message [, { transfer }])
```

Clones `message` and transmits it to `worker`'s global environment. `transfer` can be passed as a list of objects that are to be transferred rather than cloned.

The `terminate()` method, when invoked, must cause the `terminate a worker` algorithm to be run on the worker with which the object is associated.

`Worker` objects act as if they had an implicit `MessagePort` associated with them. This port is part of a channel that is set up when the worker is created, but it is not exposed. This object must never be garbage collected before the `Worker` object.

All messages received by that port must immediately be retargeted at the `Worker` object.

The `postMessage(message, transfer)` and `postMessage(message, options)` methods on `Worker` objects act as if, when invoked, they immediately invoked the respective `postMessage(message, transfer)` and `postMessage(message, options)` on the port, with the same arguments, and returned the same return value.

**Example**

The `postMessage()` method's first argument can be structured data:

```javascript
worker.postMessage({opcode: 'activate', device: 1938, parameters: [23, 102]});
```

The following are the `event handlers` (and their corresponding `event handler event types`) that must be supported, as `event handler IDL attributes`, by objects implementing the `Worker` interface:

1036
When the `Worker(scriptURL, options)` constructor is invoked, the user agent must run the following steps:

1. The user agent may throw a "SecurityError" DOMException if the request violates a policy decision (e.g. if the user agent is configured to not allow the page to start dedicated workers).
2. Let `outside settings` be the current settings object.
3. Parse the `scriptURL` argument relative to `outside settings`.
4. If this fails, throw a "SyntaxError" DOMException.
5. Let `worker URL` be the resulting URL record.
6. Let `worker` be a new `Worker` object.
7. Let `outside port` be a new `MessagePort` in `outside settings`' Realm.
8. Associate the `outside port` with `worker`.
9. Run this step in parallel:
   1. Run a worker given `worker`, `worker URL`, `outside settings`, `outside port`, and `options`.
10. Return `worker`.

10.2.6.4 Shared workers and the `SharedWorker` interface

```idl
[Exposed=Window]
interface SharedWorker : EventTarget {
    constructor(USVString scriptURL, optional (DOMString or WorkerOptions) options = {});

    readonly attribute MessagePort port;
};
SharedWorker includes AbstractWorker;
```

For web developers (non-normative)

```javascript
sharedWorker = new SharedWorker(scriptURL [, name ])
```

Returns a new `SharedWorker` object. `scriptURL` will be fetched and executed in the background, creating a new global environment for which `sharedWorker` represents the communication channel. `name` can be used to define the name of that global environment.

```javascript
sharedWorker = newSharedWorker(scriptURL [, options ])
```

Returns a new `SharedWorker` object. `scriptURL` will be fetched and executed in the background, creating a new global environment for which `sharedWorker` represents the communication channel. `options` can be used to define the name of that global environment via the `name` option. It can also ensure this new global environment supports JavaScript modules (specify `type: "module"`), and if that is specified, can also be used to specify how `scriptURL` is fetched through the `credentials` option. Note that attempting to construct a shared worker with `options` whose `type` or `credentials` values mismatch an existing shared worker will cause the returned `sharedWorker` to fire an error event and not connect to the existing shared worker.

```javascript
sharedWorker . port
```

Returns `sharedWorker`'s `MessagePort` object which can be used to communicate with the global environment.
The `port` attribute must return the value it was assigned by the object's constructor. It represents the `MessagePort` for communicating with the shared worker.

A user agent has an associated **shared worker manager** which is the result of starting a new parallel queue.

---

**Note**

Each user agent has a single shared worker manager for simplicity. Implementations could use one per origin, that would not be observably different and enables more concurrency.

---

When the `SharedWorker(scriptURL, options)` constructor is invoked:

1. Optionally, throw a "SecurityError" `DOMException` if the request violates a policy decision (e.g. if the user agent is configured to not allow the page to start shared workers).

2. If `options` is a `DOMString`, set `options` to a new `WorkerOptions` dictionary whose `name` member is set to the value of `options` and whose other members are set to their default values.

3. Let `outside settings` be the current settings object.

4. Parse `scriptURL` relative to `outside settings`.

5. If this fails, throw a "SyntaxError" `DOMException`.

6. Otherwise, let `urlRecord` be the resulting URL record.

7. Let `worker` be a new `SharedWorker` object.

8. Let `outside port` be a new `MessagePort` in `outside settings`'s `Realm`.

9. Assign `outside port` to the `port` attribute of `worker`.

10. Let `callerIsSecureContext` be true if `outside settings` is a secure context; otherwise, false.

11. Enqueue the following steps to the shared worker manager:

   1. Let `worker global scope` be null.

   2. If there exists a `SharedWorkerGlobalScope` object whose `closing` flag is false, `constructor origin` is `same origin` with `outside settings`'s `origin`, `constructor url` equals `urlRecord`, and `name` equals the value of `options`'s `name` member, then set `worker global scope` to that `SharedWorkerGlobalScope` object.

   **Note**

   data: URLs create a worker with an opaque origin. Both the `constructor origin` and `constructor url` are compared so the same data: URL can be used within an origin to get to the same `SharedWorkerGlobalScope` object, but cannot be used to bypass the same origin restriction.

   3. If `worker global scope` is not null, but the user agent has been configured to disallow communication between the worker represented by the `worker global scope` and the scripts whose `settings object` is `outside settings`, then set `worker global scope` to null.

   **Note**

   For example, a user agent could have a development mode that isolates a particular top-level browsing context from all other pages, and scripts in that development mode could be blocked from connecting to shared workers running in the normal browser mode.

   4. If `worker global scope` is not null, then check if `worker global scope`'s `type` and `credentials` match the `options` values. If not, queue a task to fire an event named `error` and abort these steps.

   5. If `worker global scope` is not null, then run these subsubsteps:

      1. Let `settings object` be the relevant settings object for `worker global scope`.

---

Any same-origin URL (including blob: URLs) can be used. data: URLs can also be used, but they create a worker with an opaque origin.
2. Let `workerIsSecureContext` be true if `settings object` is a secure context; otherwise, false.

3. If `workerIsSecureContext` is not `callerIsSecureContext`, then queue a task to fire an event named `error` at `worker` and abort these steps. [SECURE-CONTEXTS]

4. Associate `worker` with `worker global scope`.

5. Let `inside port` be a new `MessagePort` in `settings object`'s `Realm`.

6. Entangle `outside port` and `inside port`.

7. Queue a task, using the DOM manipulation task source, to fire an event named `connect` at `worker global scope`, using `MessageEvent`, with the `data` attribute initialized to the empty string, the `ports` attribute initialized to a new `frozen array` containing only `inside port`, and the `source` attribute initialized to `inside port`.

8. Append the relevant owner to add, given `outside settings` to `worker global scope`'s `owner set`.

9. Otherwise, in parallel, run a worker given `worker`, `urlRecord`, `outside settings`, `outside port`, and `options`.

10. Return `worker`.

10.2.7 Concurrent hardware capabilities

IDL

```
interface mixin NavigatorConcurrentHardware {
  readonly attribute unsigned long long hardwareConcurrency;
};
```

For web developers (non-normative)

```
self.navigator.hardwareConcurrency
```

Returns the number of logical processors potentially available to the user agent.

The `navigator.hardwareConcurrency` attribute's getter must return a number between 1 and the number of logical processors potentially available to the user agent. If this cannot be determined, the getter must return 1.

User agents should err toward exposing the number of logical processors available, using lower values only in cases where there are user-agent specific limits in place (such as a limitation on the number of `workers` that can be created) or when the user agent desires to limit fingerprinting possibilities.

10.3 APIs available to workers

10.3.1 Importing scripts and libraries

When a script invokes the `importScripts(urls)` method on a `WorkerGlobalScope` object, the user agent must import scripts into `worker global scope` given this `WorkerGlobalScope object` and `urls`.

To import scripts into worker global scope, given a `WorkerGlobalScope` object `worker global scope` and a sequence of DOMString `urls`, run these steps. The algorithm may optionally be customized by supplying custom perform the fetch hooks, which if provided will be used when invoking fetch a classic worker-imported script.

1. If `worker global scope`'s `type` is "module", throw a `TypeError` exception.

2. Let `settings object` be the `current settings object`.

3. If `urls` is empty, return.

4. Parse each value in `urls` relative to `settings object`. If any fail, throw a "SyntaxError" DOMException.

5. For each `url` in the resulting URL records, run these substeps:
1. Fetch a classic worker-imported script\(^914\) given url and settings object, passing along any custom perform the fetch\(^912\) steps provided. If this succeeds, let script be the result. Otherwise, rethrow the exception.

2. Run the classic script\(^920\) script, with the rethrow errors argument set to true.

   ![Note]
   
   script will run until it either returns, fails to parse, fails to catch an exception, or gets prematurely aborted\(^922\) by the terminate a worker\(^1034\) algorithm defined above.

   If an exception was thrown or if the script was prematurely aborted\(^922\), then abort all these steps, letting the exception or aborting continue to be processed by the calling script\(^911\).

   ![Note]
   
   Service Workers is an example of a specification that runs this algorithm with its own options for the perform the fetch\(^912\) hook. [SW]\(^1276\)

10.3.2 The **WorkerNavigator**\(^1040\) interface

The navigator attribute of the WorkerGlobalScope\(^1027\) interface must return an instance of the WorkerNavigator\(^1040\) interface, which represents the identity and state of the user agent (the client):

```
[Exposed=Worker]
interface WorkerNavigator {};
```

WorkerNavigator includes NavigatorID;
WorkerNavigator includes NavigatorLanguage;
WorkerNavigator includes NavigatorOnLine;
WorkerNavigator includes NavigatorConcurrentHardware;

10.3.3 The **WorkerLocation**\(^1040\) interface

```
[Exposed=Worker]
interface WorkerLocation {
  stringifier readonly attribute USVString href;
  readonly attribute USVString origin;
  readonly attribute USVString protocol;
  readonly attribute USVString host;
  readonly attribute USVString hostname;
  readonly attribute USVString port;
  readonly attribute USVString pathname;
  readonly attribute USVString search;
  readonly attribute USVString hash;
};
```

A WorkerLocation\(^1040\) object has an associated WorkerGlobalScope object\(^1027\) (a WorkerGlobalScope\(^1027\) object).

The href attribute's getter must return the associated WorkerGlobalScope object\(^1027\)'s url\(^1028\), serialized.

The origin attribute's getter must return the serialization\(^1040\) of the associated WorkerGlobalScope object\(^1027\)'s url\(^1028\)'s origin.

The protocol attribute's getter must return the associated WorkerGlobalScope object\(^1027\)'s url\(^1028\)'s scheme, followed by ":".

The host attribute's getter must run these steps:

1. Let url be the associated WorkerGlobalScope object\(^1027\)'s url\(^1028\).
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 hostname attribute’s getter must run these steps:

1. Let host be the associated WorkerGlobalScope object’s url’s host.
2. If host is null, return the empty string.
3. Return host, serialized.

The port attribute’s getter must run these steps:

1. Let port be the associated WorkerGlobalScope object’s url’s port.
2. If port is null, return the empty string.
3. Return port, serialized.

The pathname attribute’s getter must run these steps:

1. Let url be the associated WorkerGlobalScope object’s url.
2. If url’s cannot-be-a-base-URL is true, then return url’s path[0].
3. Return “/”, followed by the strings in url’s path (including empty strings), separated from each other by “/”.

The search attribute’s getter must run these steps:

1. Let query be the associated WorkerGlobalScope object’s url’s query.
2. If query is either null or the empty string, return the empty string.
3. Return “?”, followed by query.

The hash attribute’s getter must run these steps:

1. Let fragment be the associated WorkerGlobalScope object’s url’s fragment.
2. If fragment is either null or the empty string, return the empty string.
3. Return “#”, followed by fragment.
Worklets are a piece of specification infrastructure which can be used for running scripts independent of the main JavaScript execution environment, while not requiring any particular implementation model.

The worklet infrastructure specified here cannot be used directly by web developers. Instead, other specifications build upon it to create directly-usable worklet types, specialized for running in particular parts of the browser implementation pipeline.

Allowing extension points to rendering, or other sensitive parts of the implementation pipeline such as audio output, is difficult. If extension points were done with full access to the APIs available on Window, engines would need to abandon previously-held assumptions for what could happen in the middle of those phases. For example, during the layout phase, rendering engines assume that no DOM will be modified.

Additionally, defining extension points in the Window environment would restrict user agents to performing work in the same thread as the Window object. (Unless implementations added complex, high-overhead infrastructure to allow thread-safe APIs, as well as thread-joining guarantees.)

Worklets are designed to allow extension points, while keeping guarantees that user agents currently rely on. This is done through new global environments, based on subclasses of WorkletGlobalScope.

Worklets are similar to web workers. However, they:

• Are thread-agnostic. That is, they are not designed to run on a dedicated separate thread, like each worker is. Implementations can run worklets wherever they choose (including on the main thread).

• Are able to have multiple duplicate instances of the global scope created, for the purpose of parallelism.

• Do not use an event-based API. Instead, classes are registered on the global scope, whose methods are invoked by the user agent.

• Have a reduced API surface on the global scope.

• Have a lifetime for their global object, which is defined by other specifications, often in an implementation-defined manner.

As worklets have relatively high overhead, they are best used sparingly. Due to this, a given WorkletGlobalScope is expected to be shared between multiple separate scripts. (This is similar to how a single Window is shared between multiple separate scripts.)

Worklets are a general technology that serve different use cases. Some worklets, such as those defined in CSS Painting API, provide extension points intended for stateless, idempotent, and short-running computations, which have special considerations as described in the next couple of sections. Others, such as those defined in Web Audio API, are used for stateful, long-running operations.

Some specifications which use worklets are intended to allow user agents to parallelize work over multiple threads, or to move work between threads as required. In these specifications, user agents might invoke methods on a web-developer-provided class in an implementation-defined order.

As a result of this, to prevent interoperability issues, authors who register classes on such WorkletGlobalScope should make their
code idempotent. That is, a method or set of methods on the class should produce the same output given a particular input.

This specification uses the following techniques in order to encourage authors to write code in an idempotent way:

- No reference to the global object is available (i.e., there is no counterpart to `self` on `WorkletGlobalScope`).

  Although this was the intention when worklets were first specified, the introduction of `globalThis` has made it no longer true. See issue #6059 for more discussion.

- Code is loaded as a `module script`, which results in the code being executed in strict mode and with no shared `this` referencing the global proxy.

Together, these restrictions help prevent two different scripts from sharing state using properties of the `global object`.

Additionally, specifications which use worklets and intend to allow implementation-defined behavior must obey the following:

- They must require user agents to always have at least two `WorkletGlobalScope` instances per `Worklet`, and randomly assign a method or set of methods on a class to a particular `WorkletGlobalScope` instance. These specifications may provide an opt-out under memory constraints.

- These specifications must allow user agents to create and destroy instances of their `WorkletGlobalScope` subclasses at any time.

11.2 Examples

This section is non-normative.

For these examples, we'll use a fake worklet. The `Window` object provides two `Worklet` instances, which each run code in their own collection of `FakeWorkletGlobalScope`s:

```
IDL
partial interface Window {
    [SameObject, SecureContext] readonly attribute Worklet fakeWorklet1;
    [SameObject, SecureContext] readonly attribute Worklet fakeWorklet2;
};
```

Each `Window` has two `Worklet` instances, `fake worklet 1` and `fake worklet 2`. Both of these have their `worklet global scope type` set to `FakeWorkletGlobalScope`, and their `worklet destination type` set to "fakeworklet". User agents should create at least two `FakeWorkletGlobalScope` instances per worklet.

Note

"fakeworklet" is not actually a valid destination per Fetch. But this illustrates how real worklets would generally have their own
The `fakeWorklet1` getter steps are to return this's *fake worklet 1*.

The `fakeWorklet2` getter steps are to return this's *fake worklet 2*.

```idl
global {Worklet, FakeWorklet},
Exposed = FakeWorklet,
SecureContext
interface FakeWorkletGlobalScope {
    undefined registerFake(DOMString type, Function classConstructor);
};
```

Each `FakeWorkletGlobalScope` has a registered class constructors map, which is an ordered map, initially empty.

The `registerFake(type, classConstructor)` method steps are to set this's registered class constructors map to `classConstructor`.

### 11.2.1 Loading scripts

This section is non-normative.

To load scripts into `fake worklet 1`, a web developer would write:

```javascript
window.fakeWorklet1.addModule('script1.mjs');
window.fakeWorklet1.addModule('script2.mjs');
```

Note that which script finishes fetching and runs first is dependent on network timing: it could be either `script1.mjs` or `script2.mjs`. This generally won't matter for well-written scripts intended to be loaded in worklets, if they follow the suggestions about preparing for speculative evaluation.

If a web developer wants to perform a task only after the scripts have successfully run and loaded into some worklets, they could write:

```javascript
Promise.all([window.fakeWorklet1.addModule('script1.mjs'),
             window.fakeWorklet2.addModule('script2.mjs')]).then(() => {
    // Do something which relies on those scripts being loaded.
});
```

Another important point about script-loading is that loaded scripts can be run in multiple `WorkletGlobalScope`s per `Worklet`, as discussed in the section on code idempotence. In particular, the specification above for `fake worklet 1` and `fake worklet 2` require this. So, consider a scenario such as the following:

```javascript
// script.mjs
console.log("Hello from a FakeWorkletGlobalScope!");

// app.mjs
window.fakeWorklet1.addModule("script.mjs");
```

This could result in output such as the following from a user agent's console:

```
[fakeWorklet1#1] Hello from a FakeWorkletGlobalScope!
[fakeWorklet1#4] Hello from a FakeWorkletGlobalScope!
```
If the user agent at some point decided to kill and restart the third instance of `FakeWorkletGlobalScope`, the console would again print `Hello from a FakeWorkletGlobalScope!` when this occurs.

### 11.2.2 Registering a class and invoking its methods

This section is non-normative.

Let’s say that one of the intended usages of our fake worklet by web developers is to allow them to customize the highly-complex process of boolean negation. They might register their customization as follows:

```javascript
// script.mjs
registerFake('negation-processor', class {
  process(arg) {
    return !arg;
  }
});

// app.mjs
window.fakeWorklet1.addModule("script.mjs");
```

To make use of such registered classes, the specification for fake worklets could define a **find the opposite of true** algorithm, given a `Worklet` worklet:

1. Optionally, create a `worklet global scope` for `worklet`.
2. Let `workletGlobalScope` be one of `worklet`'s `global scopes`, chosen in an implementation-defined manner.
3. Let `classConstructor` be `workletGlobalScope`'s `registered class constructors map`[`"negation-processor"`].
4. Let `classInstance` be the result of constructing `classConstructor`, with no arguments.
5. Let `function` be `Get(classInstance, "process")`. Rethrow any exceptions.
6. Let `callback` be the result of converting `function` to a Web IDL `Function` instance.
7. Return the result of invoking `callback` with the arguments « true » and with `classInstance` as the `callback this value`.

**Note**

Another, perhaps better, specification architecture would be to extract the "process" property and convert it into a `Function` at registration time, as part of the `registerFake()` method steps.

### 11.3 Infrastructure

#### 11.3.1 The global scope

Subclasses of `WorkletGlobalScope` are used to create `global objects` wherein code loaded into a particular `Worklet` can execute.

```idl
[Exposed=Worklet, SecureContext]
interface WorkletGlobalScope {};
```

**Note**

Other specifications are intended to subclass `WorkletGlobalScope`, adding APIs to register a class, as well as other APIs specific for their worklet type.
Each WorkletGlobalScope has an associated module map. It is a module map, initially empty.

### 11.3.1.1 Agents and event loops

This section is non-normative.

Each WorkletGlobalScope is contained in its own worklet agent, which has its corresponding event loop. However, in practice, implementation of these agents and event loops is expected to be different from most others.

A worklet agent exists for each WorkletGlobalScope instance, since, in theory, an implementation could use a separate thread for each WorkletGlobalScope instance, and allowing this level of parallelism is best done using agents. However, because their [[CanBlock]] value is false, there is no requirement that agents and threads are one-to-one. This allows implementations the freedom to execute scripts loaded into a worklet on any thread, including one running code from other agents with [[CanBlock]] of false, such as the thread of a similar-origin window agent ("the main thread"). Contrast this with dedicated worker agents, whose true value for [[CanBlock]] effectively requires them to get a dedicated operating system thread.

Worklet event loops are also somewhat special. They are only used for tasks associated with addModule(), tasks wherein the user agent invokes author-defined methods, and microtasks. Thus, even though the event loop processing model specifies that all event loops run continuously, implementations can achieve observably-equivalent results using a simpler strategy, which just invokes author-provided methods and then relies on that process to perform a microtask checkpoint.

### 11.3.1.2 Creation and termination

To create a worklet global scope for a Worklet worklet:

1. Let outsideSettings be worklet’s relevant settings object.
2. Let agent be the result of obtaining a worklet agent given outsideSettings. Run the rest of these steps in that agent.
3. Let realmExecutionContext be the result of creating a new JavaScript realm given agent and the following customizations:
   - For the global object, create a new object of the type given by worklet’s worklet global scope type.
4. Let workletGlobalScope be the global object of realmExecutionContext’s Realm component.
5. Let insideSettings be the result of setting up a worklet environment settings object given realmExecutionContext and outsideSettings.
6. Initialize a global object’s CSP list given workletGlobalScope. [CSP]
7. For each moduleURL of worklet’s added modules list:
   1. Fetch a worklet script graph given moduleURL, insideSettings, worklet’s worklet destination type, what credentials mode?, insideSettings, and worklet’s module responses map. Wait until the algorithm asynchronously completes with script.

   **Note**
   This will not actually perform a network request, as it will just reuse responses from worklet’s module responses map. The main purpose of this step is to create a new workletGlobalScope-specific module script from the response.

   2. Assert: script is not null, since the fetch succeeded and the source text was successfully parsed when worklet’s module responses map was initially populated with moduleURL.
3. Run a module script given script.
8. Append workletGlobalScope to outsideSettings’s global object’s associated Document’s worklet global scopes.
9. Append workletGlobalScope to worklet’s global scopes.
10. Run the responsible event loop specified by insideSettings.

To terminate a worklet global scope given a WorkletGlobalScope workletGlobalScope:
1. Let `eventLoop` be `workletGlobalScope`'s relevant agent's event loop.

2. If there are any tasks queued in `eventLoop`'s task queues, discard them without processing them.

3. Wait for `eventLoop` to complete the currently running task.

4. If the previous step doesn't complete within an implementation-defined period of time, then abort the script currently running in the worklet.

5. Destroy `eventLoop`.

6. Remove `workletGlobalScope` from the global scopes of the `Worklet` whose global scopes contains `workletGlobalScope`.

7. Remove `workletGlobalScope` from the worklet global scopes of the `Document` whose worklet global scopes contains `workletGlobalScope`.

11.3.1.3 Script settings for worklets

To set up a worklet environment settings object, given a JavaScript execution context, `executionContext` and an environment settings object outsideSettings:

1. Let `origin` be a unique opaque origin.

2. Let `inheritedAPIBaseURL` be outsideSettings's API base URL.

3. Let `inheritedReferrerPolicy` be outsideSettings's referrer policy.

4. Let `inheritedEmbedderPolicy` be outsideSettings's embedder policy.

5. Let `realm` be the value of `executionContext`'s Realm component.

6. Let `workletGlobalScope` be `realm`'s global object.

7. Let `settingsObject` be a new environment settings object whose algorithms are defined as follows:

   **The `realm execution context`**
   - Return `executionContext`.

   **The `module map`**
   - Return `workletGlobalScope`'s module map.

   **The `responsible document`**
   - Not applicable (the responsible event loop is not a window event loop).

   **The `API URL character encoding`**
   - Return UTF-8.

   **The `API base URL`**
   - Return `inheritedAPIBaseURL`.

   **Note**
   Unlike workers or other globals derived from a single resource, worklets have no primary resource; instead, multiple scripts, each with their own URL, are loaded into the global scope via `worklet.addModule()`. So this API base URL is rather unlike that of other globals. However, so far this doesn't matter, as no APIs available to worklet code make use of the API base URL.

   **The `origin`**
   - Return `origin`.

   **The `referrer policy`**
   - Return `inheritedReferrerPolicy`.
The embedder policy
Return inheritedEmbedderPolicy.

The cross-origin isolated capability
Return TODO.

8. Set settingsObject's id to a new unique opaque string, creation URL to inheritedAPIBaseURL, top-level creation URL to null, top-level origin to outsideSettings's top-level origin, target browsing context to null, and active service worker to null.

9. Set realm's [[HostDefined]] field to settingsObject.


11.3.2 The Worklet class

The Worklet class provides the capability to add module scripts into its associated WorkletGlobalScope s. The user agent can then create classes registered on the WorkletGlobalScope s and invoke their methods.

```
IDL  [Exposed=Window, SecureContext]
interface Worklet {
    [NewObject] Promise<undefined> addModule(USVString moduleURL, optional WorkletOptions options = {});
};

dictionary WorkletOptions {
    RequestCredentials credentials = "same-origin";
};
```

Specifications that create Worklet instances must specify the following for a given instance:

- its worklet global scope type, which must be a Web IDL type that inherits from WorkletGlobalScope; and
- its worklet destination type, which must be a destination, and is used when fetching scripts.

For web developers (non-normative)

```
await worklet.addModule(moduleURL[, { credentials }])
```

Loads and executes the module script given by moduleURL into all of worklet's global scopes. It can also create additional global scopes as part of this process, depending on the worklet type. The returned promise will fulfill once the script has been successfully loaded and run in all global scopes.

The credentials option can be set to a credentials mode to modify the script-fetching process. It defaults to "same-origin".

Any failures in fetching the script or its dependencies will cause the returned promise to be rejected with an "AbortError" DOMException. Any errors in parsing the script or its dependencies will cause the returned promise to be rejected with the exception generated during parsing.

A Worklet has a list of global scopes, which contains instances of the Worklet's worklet global scope. It is initially empty.

A Worklet has an added modules list, which is a list of URLs, initially empty. Access to this list should be thread-safe.

A Worklet has a module responses map, which is an ordered map from URLs to responses, initially empty. Access to this map should be thread-safe.

Note

The added modules list and module responses map exist to ensure that WorkletGlobalScope s created at different times get equivalent module scripts run in them, based on the same source text. This allows the creation of additional WorkletGlobalScope s to be transparent to the author.
In practice, user agents are not expected to implement these data structures, and the algorithms that consult them, using thread-safe programming techniques. Instead, when `addModule()` is called, user agents can fetch the module graph on the main thread, and send the fetched source text (i.e., the important data contained in the `module responses map`) to each thread which has a `WorkletGlobalScope`.

Then, when a user agent creates a new `WorkletGlobalScope` for a given `Worklet`, it can simply send the map of fetched source text and the list of entry points from the main thread to the thread containing the new `WorkletGlobalScope`.

The `addModule(moduleURL, options)` method steps are:

1. Let `outsideSettings` be the relevant settings object of `this`.
2. Parse `moduleURL` relative to `outsideSettings`.
3. If this fails, then return a promise rejected with a `SyntaxError` DOMException.
4. Let `moduleURLRecord` be the resulting URL record.
5. Let `promise` be a new promise.
6. Run the following steps in parallel:
   1. If `this`'s global scopes is empty, then:
      1. Create a worklet global scope given `this`.
      2. Optionally, create additional global scope instances given `this`, depending on the specific worklet in question and its specification.
      3. Wait for all steps of the creation process(es) — including those taking place within the worklet agents — to complete, before moving on.
   2. Let `pendingTasks` be `this`'s global scopes's size.
   3. Let `addedSuccessfully` be false.
   4. For each `workletGlobalScope` of `this`'s global scopes, queue a global task on the networking task source given `workletGlobalScope` to perform the following steps:
      1. Fetch a worklet script graph given `moduleURLRecord`, `outsideSettings`, `this`'s worklet destination type, `options`, `workletGlobalScope`'s relevant settings object, and `this`'s `module responses map`. Wait until the algorithm asynchronously completes with `script`.
      2. If `script` is null, then:
         1. Queue a global task on the networking task source given `this`'s relevant global object to perform the following steps:
            1. If `pendingTasks` is not −1, then:
               1. Set `pendingTasks` to −1.
               2. Reject `promise` with an "AbortError" DOMException.
            2. Abort these steps.
      3. If `script`'s `error to rethrow` is not null, then:
         1. Queue a global task on the networking task source given `this`'s relevant global object to perform the following steps:
            1. If `pendingTasks` is not −1, then:
1. Set `pendingTasks` to −1.
2. Reject `promise` with script’s error to rethrow.
3. Abort these steps.
4. If `addedSuccessfully` is false, then:
   1. Append `moduleURLRecord` to this’s `added modules list`.
   2. Set `addedSuccessfully` to true.
5. Run a module script given script.
6. Queue a global task on the networking task source given this’s relevant global object to perform the following steps:
   1. If `pendingTasks` is not −1, then:
      1. Set `pendingTasks` to `pendingTasks` − 1.
      2. If `pendingTasks` is 0, then resolve `promise`.
7. Return `promise`.

### 11.3.3 The worklet’s lifetime

The lifetime of a `Worklet` has no special considerations; it is tied to the object it belongs to, such as the `Window`.

Each `Document` has a **worklet global scopes**, which is a set of `WorkletGlobalScope` s, initially empty.

The lifetime of a `WorkletGlobalScope` is, at a minimum, tied to the `Document` whose `worklet global scopes` contain it. In particular, discarding the `Document` will terminate the corresponding `WorkletGlobalScope` and allow it to be garbage-collected.

Additionally, user agents may, at any time, terminate a given `WorkletGlobalScope`, unless the specification defining the corresponding worklet type says otherwise. For example, they might terminate them if the `worklet agent’s event loop` has no tasks queued, or if the user agent has no pending operations planning to make use of the worklet, or if the user agent detects abnormal operations such as infinite loops or callbacks exceeding imposed time limits.

Finally, specifications for specific worklet types can give more specific details on when to create `WorkletGlobalScope` s for a given worklet type. For example, they might create them during specific processes that call upon worklet code, as in the example.
12 Web storage

12.1 Introduction

This section is non-normative.

This specification introduces two related mechanisms, similar to HTTP session cookies, for storing name-value pairs on the client side.

The first is designed for scenarios where the user is carrying out a single transaction, but could be carrying out multiple transactions in different windows at the same time.

Cookies don't really handle this case well. For example, a user could be buying plane tickets in two different windows, using the same site. If the site used cookies to keep track of which ticket the user was buying, then as the user clicked from page to page in both windows, the ticket currently being purchased would "leak" from one window to the other, potentially causing the user to buy two tickets for the same flight without really noticing.

To address this, this specification introduces the `sessionStorage` getter. Sites can add data to the session storage, and it will be accessible to any page from the same site opened in that window.

**Example**

For example, a page could have a checkbox that the user ticks to indicate that they want insurance:

```html
<label>
  <input type="checkbox" onchange="sessionStorage.insurance = checked ? 'true' : 'false'"> I want insurance on this trip.
</label>
```

A later page could then check, from script, whether the user had checked the checkbox or not:

```javascript
if (sessionStorage.insurance) {
  ...
}
```

If the user had multiple windows opened on the site, each one would have its own individual copy of the session storage object.

The second storage mechanism is designed for storage that spans multiple windows, and lasts beyond the current session. In particular, web applications might wish to store megabytes of user data, such as entire user-authored documents or a user's mailbox, on the client side for performance reasons.

Again, cookies do not handle this case well, because they are transmitted with every request.

The `localStorage` getter is used to access a page's local storage area.

**Example**

The site at example.com can display a count of how many times the user has loaded its page by putting the following at the bottom of its page:

```html
<p>
  You have viewed this page
  <span id="count">an untold number of</span> time(s).
</p>
<script>
  if (!localStorage.pageLoadCount) {
    localStorage.pageLoadCount = 0;
  }
</script>
```
Each site has its own separate storage area.

⚠ Warning! The `localStorage` getter provides access to shared state. This specification does not define the interaction with other browsing contexts in a multiprocess user agent, and authors are encouraged to assume that there is no locking mechanism. A site could, for instance, try to read the value of a key, increment its value, then write it back out, using the new value 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.

12.2 The API

12.2.1 The `Storage` interface

```idl
[Exposed=Window]
interface Storage {
    readonly attribute unsigned long length;
    DOMString? key(unsigned long index);
    getter DOMString? getItem(DOMString key);
    setter undefined setItem(DOMString key, DOMString value);
    deleter undefined removeItem(DOMString key);
    undefined clear();
};
```

For web developers (non-normative)

- `storage.length` Returns the number of key/value pairs.
- `storage.key(n)` Returns the name of the nth key, or null if n is greater than or equal to the number of key/value pairs.
- `value = storage.getItem(key)`
- `value = storage[key]` Returns the current value associated with the given key, or null if the given key does not exist.
- `storage.setItem(key, value)` Sets the value of the pair identified by `key` to `value`, creating a new key/value pair if none existed for `key` previously.
  - Throws a "QuotaExceededError" DOMException exception if the new value couldn't be set. (Setting could fail if, e.g., the user has disabled storage for the site, or if the quota has been exceeded.)
  - Dispatches a `storage` event on `Window` objects holding an equivalent `Storage` object.
- `storage.removeItem(key)` Removes the key/value pair with the given `key`, if a key/value pair with the given `key` exists.
  - Dispatches a `storage` event on `Window` objects holding an equivalent `Storage` object.
A Storage object has an associated:

- **map**: A storage proxy map.
- **type**: "local" or "session".

To reorder a Storage object storage, reorder storage’s map’s entries in an implementation-defined manner.

**Note**

Unfortunate as it is, iteration order is not defined and can change upon most mutations.

To broadcast a Storage object storage, given a key, oldValue, and newValue, run these steps:

1. Let url be storage’s relevant global object’s associated Document’s URL.
2. Let remoteStorages be all Storage objects excluding storage whose:
   - type is storage’s type
   - relevant settings object’s origin is same origin with storage’s relevant settings object’s origin.
   and, if type is "session", whose relevant settings object’s browsing session is storage’s relevant settings object’s browsing session.
3. For each remoteStorage of remoteStorages: queue a global task on the DOM manipulation task source given remoteStorage’s relevant global object’s to fire an event named storage at remoteStorage’s relevant global object, using StorageEvent, with key initialized to key, oldValue initialized to oldValue, newValue initialized to newValue, url initialized to url, and storageArea initialized to remoteStorage.

**Note**

The Document object associated with the resulting task is not necessarily fully active, but events fired on such objects are ignored by the event loop until the Document becomes fully active again.

The length getter steps are to return this’s map’s size.

The key method steps are:

1. If n is greater than or equal to this’s map’s size, then return null.
2. Let keys be the result of running get the keys on this’s map.
3. Return keys[n].

The supported property names on a Storage object storage are the result of running get the keys on storage’s map.

The getItem method steps are:

1. If this’s map[key] does not exist, then return null.
2. Return this’s map[key].

The setItem(key, value) method are:

1. Let oldValue be null.
2. Let reorder be true.

The clear() method removes all key/value pairs, if there are any.

Dispatches a storage event on Window objects holding an equivalent Storage object.
3. If this's map\(^{1053}\)[key] exists:
   1. Set oldValue to this's map\(^{1053}\)[key].
   2. If oldValue is value, then return.
   3. Set reorder to false.
4. If value cannot be stored, then throw a "QuotaExceededError" DOMException exception.
5. Set this's map\(^{1053}\)[key] to value.
6. If reorder is true, then reorder this.
7. Broadcast this with key, oldValue, and value.

The removeItem(key) method steps are:
1. If this's map\(^{1053}\)[key] does not exist, then return null.
2. Set oldValue to this's map\(^{1053}\)[key].
3. Remove this's map\(^{1053}\)[key].
4. Reorder this.
5. Broadcast this with key, oldValue, and null.

The clear() method steps are:
1. Clear this's map.
2. Broadcast this with null, null, and null.

12.2.2 The sessionStorage getter

IDL
interface mixin WindowSessionStorage {
   readonly attribute Storage sessionStorage;
};
Window includes WindowSessionStorage;

For web developers (non-normative)

window . sessionStorage

Returns the Storage object associated with that window's origin's session storage area.

Throws a "SecurityError" DOMException if the Document's origin is an opaque origin or if the request violates a policy decision (e.g., if the user agent is configured to not allow the page to persist data).

A Document object has an associated session storage holder, which is null or a Storage object. It is initially null.

The sessionStorage getter steps are:
1. If this's associated Document's session storage holder is non-null, then return this's associated Document's session storage holder.
2. Let map be the result of running obtain a session storage bottle map with this's relevant settings object and "sessionStorage".
3. If map is failure, then throw a "SecurityError" DOMException.
4. Let storage be a new Storage object whose map is map.
5. Set this's associated Document's session storage holder to storage.
6. Return storage.
12.2.3 The `localStorage` getter

IDL

```idl
interface mixin WindowLocalStorage {
  readonly attribute Storage localStorage;
};
Window includes WindowLocalStorage;
```

For web developers (non-normative)

```javascript
window.localStorage
```

Returns the `Storage` object associated with `window`'s origin's local storage area.

 Throws a "SecurityError" DOMException if the `Document`'s `origin` is an opaque origin or if the request violates a policy decision (e.g., if the user agent is configured to not allow the page to persist data). A `Document` object has an associated local storage holder, which is null or a `Storage` object. It is initially null.

The `localStorage` getter steps are:

1. If this's associated `Document`'s local storage holder is non-null, then return this's associated `Document`'s local storage holder.
2. Let `map` be the result of running `obtain a local storage bottle map` with this's relevant settings object and "localStorage".
3. If `map` is failure, then throw a "SecurityError" DOMException.
4. Let `storage` be a new `Storage` object whose `map` is `map`.
5. Set this's associated `Document`'s local storage holder to `storage`.
6. Return `storage`.

12.2.4 The `StorageEvent` interface

IDL

```idl
[Exposed=Window]
interface StorageEvent : Event {
  constructor(DOMString type, optional StorageEventInit eventInitDict = {});
}
```

```javascript
readonly attribute DOMString? key;
readonly attribute DOMString? oldValue;
readonly attribute DOMString? newValue;
readonly attribute USVString url;
readonly attribute Storage? storageArea;
```

```javascript
undefined initStorageEvent(DOMString type, optional boolean bubbles = false, optional boolean cancelable = false, optional DOMString? key = null, optional DOMString? oldValue = null, optional DOMString? newValue = null, optional USVString url = "", optional Storage? storageArea = null);
```

```javascript
dictionary StorageEventInit : EventInit {
  DOMString? key = null;
  DOMString? oldValue = null;
  DOMString? newValue = null;
  USVString url = "";
  Storage? storageArea = null;
}
```
The `key`, `oldValue`, `newValue`, `url`, and `storageArea` attributes must return the values they were initialized to.

The `initStorageEvent()` method must initialize the event in a manner analogous to the similarly-named `initEvent()` method.

### 12.3 Privacy

#### 12.3.1 User tracking

A third-party advertiser (or any entity capable of getting content distributed to multiple sites) could use a unique identifier stored in its local storage area to track a user across multiple sessions, building a profile of the user's interests to allow for highly targeted advertising. In conjunction with a site that is aware of the user's real identity (for example an e-commerce site that requires authenticated credentials), this could allow oppressive groups to target individuals with greater accuracy than in a world with purely anonymous web usage.

There are a number of techniques that can be used to mitigate the risk of user tracking:

**Blocking third-party storage**

User agents may restrict access to the `LocalStorage` objects to scripts originating at the domain of the active document of the top-level browsing context, for instance denying access to the API for pages from other domains running in `iframe`s.

**Expiring stored data**

User agents may, possibly in a manner configured by the user, automatically delete stored data after a period of time.

For example, a user agent could be configured to treat third-party local storage areas as session-only storage, deleting the data once the user had closed all the browsing contexts that could access it.

This can restrict the ability of a site to track a user, as the site would then only be able to track the user across multiple sessions when they authenticate with the site itself (e.g. by making a purchase or logging in to a service).

However, this also reduces the usefulness of the API as a long-term storage mechanism. It can also put the user's data at risk, if the user does not fully understand the implications of data expiration.

**Treating persistent storage as cookies**

If users attempt to protect their privacy by clearing cookies without also clearing data stored in the local storage area, sites can defeat those attempts by using the two features as redundant backup for each other. User agents should present the interfaces for clearing these in a way that helps users to understand this possibility and enables them to delete data in all persistent storage features simultaneously.

[COOKIES]

**Site-specific safelisting of access to local storage areas**

User agents may allow sites to access session storage areas in an unrestricted manner, but require the user to authorize access to local storage areas.

---

For web developers (non-normative)

- `event.key`
  - Returns the key of the storage item being changed.

- `event.oldValue`
  - Returns the old value of the key of the storage item whose value is being changed.

- `event.newValue`
  - Returns the new value of the key of the storage item whose value is being changed.

- `event.url`
  - Returns the URL of the document whose storage item changed.

- `event.storageArea`
  - Returns the `Storage` object that was affected.
Origin-tracking of stored data

User agents may record the origins of sites that contained content from third-party origins that caused data to be stored.

If this information is then used to present the view of data currently in persistent storage, it would allow the user to make informed decisions about which parts of the persistent storage to prune. Combined with a blocklist ("delete this data and prevent this domain from ever storing data again"), the user can restrict the use of persistent storage to sites that they trust.

Shared blocklists

User agents may allow users to share their persistent storage domain blocklists.

This would allow communities to act together to protect their privacy.

While these suggestions prevent trivial use of this API for user tracking, they do not block it altogether. Within a single domain, a site can continue to track the user during a session, and can then pass all this information to the third party along with any identifying information (names, credit card numbers, addresses) obtained by the site. If a third party cooperates with multiple sites to obtain such information, a profile can still be created.

However, user tracking is to some extent possible even with no cooperation from the user agent whatsoever, for instance by using session identifiers in URLs, a technique already commonly used for innocuous purposes but easily repurposed for user tracking (even retroactively). This information can then be shared with other sites, using visitors’ IP addresses and other user-specific data (e.g. user-agent headers and configuration settings) to combine separate sessions into coherent user profiles.

12.3.2 Sensitivity of data

User agents should treat persistently stored data as potentially sensitive; it’s quite possible for emails, calendar appointments, health records, or other confidential documents to be stored in this mechanism.

To this end, user agents should ensure that when deleting data, it is promptly deleted from the underlying storage.

12.4 Security

12.4.1 DNS spoofing attacks

Because of the potential for DNS spoofing attacks, one cannot guarantee that a host claiming to be in a certain domain really is from that domain. To mitigate this, pages can use TLS. Pages using TLS can be sure that only the user, software working on behalf of the user, and other pages using TLS that have certificates identifying them as being from the same domain, can access their storage areas.

12.4.2 Cross-directory attacks

Different authors sharing one host name, for example users hosting content on the now defunct geocities.com, all share one local storage object. There is no feature to restrict the access by pathname. Authors on shared hosts are therefore urged to avoid using these features, as it would be trivial for other authors to read the data and overwrite it.

Note

Even if a path-restriction feature was made available, the usual DOM scripting security model would make it trivial to bypass this protection and access the data from any path.

12.4.3 Implementation risks

The two primary risks when implementing these persistent storage features are letting hostile sites read information from other domains, and letting hostile sites write information that is then read from other domains.

Letting third-party sites read data that is not supposed to be read from their domain causes information leakage. For example, a user’s
shopping wishlist on one domain could be used by another domain for targeted advertising; or a user's work-in-progress confidential documents stored by a word-processing site could be examined by the site of a competing company.

Letting third-party sites write data to the persistent storage of other domains can result in *information spoofing*, which is equally dangerous. For example, a hostile site could add items to a user's wishlist; or a hostile site could set a user's session identifier to a known ID that the hostile site can then use to track the user's actions on the victim site.

Thus, strictly following the [origin model](#) described in this specification is important for user security.
13 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 XML syntax”.

13.1 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 ASCII whitespace.
3. A DOCTYPE.
4. Any number of comments and ASCII whitespace.
5. The document element, in the form of an html element.
6. Any number of comments and ASCII whitespace.

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.

Note
ASCII whitespace before the html element, at the start of the html element and before the head element, will be dropped when the document is parsed; ASCII whitespace after the html element will be parsed as if it were at the end of the body element. Thus, ASCII whitespace around the document element does not round-trip.

It is suggested that newlines be inserted after the DOCTYPE, after any comments that are before the document 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 ASCII upper alphas and ASCII lower alphas. For convenience, in this section this is just referred to as “case-insensitive”.

13.1.1 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 ASCII whitespace.
3. A string that is an ASCII case-insensitive match for the string "html".
4. Optionally, a DOCTYPE legacy string.
5. Zero or more ASCII whitespace.
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 ASCII whitespace.
2. A string that is an ASCII case-insensitive match for the string "SYSTEM".
3. One or more ASCII whitespace.
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.

13.1.2 Elements

There are six different kinds of elements: void elements, the template element, raw text elements, escapable raw text elements, foreign elements, and normal elements.

**Void elements**

area, base, br, col, embed, hr, img, input, link, meta, param, source, track, wbr

**The template element**

template

**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).
The template element can have template contents, but such template contents are not children of the template element itself. Instead, they are stored in a DocumentFragment associated with a different Document — without a browsing context — so as to avoid the template contents interfering with the main Document. The markup for the template contents of a template element is placed just after the template element's start tag and just before template element's end tag (as with other elements), and may consist of any text, character references, elements, and comments, but the text must not contain the character U+003C LESS-THAN SIGN (<) or an ambiguous ampersand.

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.

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.

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 ASCII alphanumerics. 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.

13.1.2.1 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 ASCII whitespace.
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 ASCII whitespace.
5. After the attributes, or after the tag name if there are no attributes, there may be one or more ASCII whitespace. (Some attributes are required to be followed by a space. See the attributes section 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

Note
The HTML syntax does not support namespace declarations, even in foreign elements.

For instance, consider the following HTML fragment:

```html
<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 SVG 2 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.
start tag as self-closing.

7. Finally, start tags must be closed by a U+003E GREATER-THEAN SIGN character (>).

13.1.2.2 End tags

End tags must have the following format:

1. The first character of an end tag must be a U+003C LESS-THEAN 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 ASCII whitespace.
5. Finally, end tags must be closed by a U+003E GREATER-THEAN SIGN character (>).

13.1.2.3 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 controls, U+0020 SPACE, U+0022 QUOTATION MARK, U+0027 APOSTROPHE, U+003D EQUALS SIGN, U+003C LESS-THEAN SIGN, U+003E GREATER-THEAN SIGN, and noncharacters. In the HTML syntax, attribute names, even those for foreign elements, may be written with any mix of ASCII lower and ASCII upper alphas.

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
In the following example, the disabled attribute is given with the empty attribute syntax:

```html
<input disabled>
```

If an attribute using the empty attribute syntax is to be followed by another attribute, then there must be ASCII whitespace separating the two.

Unquoted attribute value syntax

The attribute name, followed by zero or more ASCII whitespace, followed by a single U+003D_EQUALS SIGN character, followed by zero or more ASCII whitespace, followed by the attribute value, which, in addition to the requirements given above for attribute values, must not contain any literal ASCII whitespace, any U+0022 QUOTATION MARK characters ("), U+0027 APOSTROPHE characters ('), U+003D_EQUALS SIGN characters (=), U+003C_LESS-THEAN SIGN characters (<), U+003E_GREATER-THEAN SIGN characters (>), or U+0060 GRAVE Accent characters (`), and must not be the empty string.

Example
In the following example, the value attribute is given with the unquoted attribute value syntax:

```html
<input value=yes>
```

If an attribute using the unquoted attribute value 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 ASCII whitespace separating the two.
**Single-quoted attribute value syntax**

The attribute name, followed by zero or more ASCII whitespace, followed by a single U+003D EQUALS SIGN character, followed by zero or more ASCII whitespace, 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**

In the following example, the type attribute is given with the single-quoted attribute value syntax:

```html
<input type='checkbox'>
```

If an attribute using the single-quoted attribute syntax is to be followed by another attribute, then there must be ASCII whitespace separating the two.

**Double-quoted attribute value syntax**

The attribute name, followed by zero or more ASCII whitespace, followed by a single U+003D EQUALS SIGN character, followed by zero or more ASCII whitespace, 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**

In the following example, the name attribute is given with the double-quoted attribute value syntax:

```html
<input name="be evil">
```

If an attribute using the double-quoted attribute syntax is to be followed by another attribute, then there must be ASCII whitespace 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 named 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.

<table>
<thead>
<tr>
<th>Local name</th>
<th>Namespace</th>
<th>Attribute name</th>
</tr>
</thead>
<tbody>
<tr>
<td>actuate</td>
<td>XLink namespace</td>
<td>xlink:actuate</td>
</tr>
<tr>
<td>arcrole</td>
<td>XLink namespace</td>
<td>xlink:arcrole</td>
</tr>
<tr>
<td>href</td>
<td>XLink namespace</td>
<td>xlink:href</td>
</tr>
<tr>
<td>role</td>
<td>XLink namespace</td>
<td>xlink:role</td>
</tr>
<tr>
<td>show</td>
<td>XLink namespace</td>
<td>xlink:show</td>
</tr>
<tr>
<td>title</td>
<td>XLink namespace</td>
<td>xlink:title</td>
</tr>
<tr>
<td>type</td>
<td>XLink namespace</td>
<td>xlink:type</td>
</tr>
<tr>
<td>lang</td>
<td>XML namespace</td>
<td>xml:lang</td>
</tr>
<tr>
<td>space</td>
<td>XML namespace</td>
<td>xml:space</td>
</tr>
<tr>
<td>xmlns</td>
<td>XMLNS namespace</td>
<td>xmlns</td>
</tr>
<tr>
<td>xlink</td>
<td>XMLNS namespace</td>
<td>xmlns:xlink</td>
</tr>
</tbody>
</table>

No other named attributes 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. SVG 2 and MathML); this section only describes the syntax rules if the attributes are serialized using the HTML syntax.

**13.1.2.4 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**

For example, in the following case it's ok to remove the "<html>" tag:

```html
<!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:

```html
<!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 whitespace around the document element is ignored by the parser. The following example would also have the exact same DOM:

```html
<!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:

```html
<!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:

```html
<!DOCTYPE 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 ASCII whitespace 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 ASCII whitespace 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

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 whitespace:

```
<!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, whitespace isn't an issue. If we first remove the whitespace 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 whitespace back:

```
<!DOCTYPE HTML>
<title>Hello</title>
```
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>` or `<h6>` element, or if there is no more content in the parent element and the parent element is an `<HTML element>` that is not an `<audio>`, `<del>`, `<ins>`, `<map>`, `<noscript>`, `<video>` element, or an autonomous custom element.

**Example**

We can thus simplify the earlier example further:

```html
<!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 ASCII whitespace or a `<comment>`.

A `<caption>` element’s `<end tag>` may be omitted if the `<caption>` element is not immediately followed by ASCII whitespace 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 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.

The ability to omit all these table-related tags makes table markup much terser.

The exact same table, modulo some whitespace differences, could be marked up as follows:

```xml
<caption>37547 TEE Electric Powered Rail Car Train Functions (Abbreviated)</caption>
<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>
```
However, a **start tag** must never be omitted if it has any attributes.

Since the cells take up much less room this way, this can be made even terser by having each row on one line:

```html
<table>
  <caption>37547 TEE Electric Powered Rail Car Train Functions (Abbreviated)</caption>
  <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 only differences between these tables, at the DOM level, is with the precise position of the (in any case semantically-neutral) whitespace.

**However**, a **start tag** must never be omitted if it has any attributes.

**Example**

Returning to the earlier example with all the whitespace removed and then all the optional tags removed:

```html
<!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:

```html
<!DOCTYPE HTML><html lang="en"><title>Hello</title><body class="demo"><p>Welcome to this example.
```
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).

The following two pre blocks are equivalent:

```
<pre>Hello</pre>

<pre>Hello</pre>
```

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 (/).

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.

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.

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 the 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 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 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 code point excluding U+000D CR, noncharacters, and controls other than ASCII whitespace.

An ambiguous ampersand is a U+0026 AMPERSAND character (&) that is followed by one or more ASCII alphanumerics, followed by a U+003B SEMICOLON character (;), where these characters do not match any of the names given in the named character references section.

13.1.5 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

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 a MathML ms element:

```xml
<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>
```

13.1.6 Comments

Comments must have the following format:

1. The string "<!--".
2. Optionally, text, with the additional restriction that the text must not start with the string ">", nor start with the string "-->", nor contain the strings "<!--", "-->", or "--!>", nor end with the string "<!-".
3. The string "-->".

Note

The text is allowed to end with the string "<!", as in <!--My favorite operators are > and <!-->.
This section only applies to user agents, data mining tools, and conformance checkers.

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**.

### Note

The rules for parsing XML documents into DOM trees are covered by the next section, entitled "The XML 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**.

### Note

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 HTML2 to HTML4) 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.

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 **menu** element", then that is an element with the local name "menu", the namespace "http://www.w3.org/1999/xhtml", and the interface **HTMLMenuElement**. Where possible, references to such elements are hyperlinked to their definition.
The input to the HTML parsing process consists of a stream of code points, which is passed through a tokenization stage followed by a tree construction stage. The output is a Document object.

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**

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:

```html
...<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.

### 13.2.2 Parse errors

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.

Some parse errors have dedicated codes outlined in the table below that should be used by conformance checkers in reports.

**Error descriptions in the table below are non-normative.**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abrupt-closing-of-empty-comment</td>
<td>This error occurs if the parser encounters an empty comment that is abruptly closed by a U+003E (&gt;) code point (i.e., &lt;!- - or &lt;!- -&gt;). The parser behaves as if the comment is closed correctly.</td>
</tr>
<tr>
<td>abrupt-doctype-public-identifier</td>
<td>This error occurs if the parser encounters a U+003E (&gt;) code point in the DOCTYPE public identifier (e.g., &lt;!DOCTYPE html PUBLIC &quot;foo&quot;&gt;). In such a case, if the doctype is correctly placed as a document preamble, the parser sets the document to quirks mode.</td>
</tr>
<tr>
<td>abrupt-doctype-system-identifier</td>
<td>This error occurs if the parser encounters a U+003E (&gt;) code point in the DOCTYPE system identifier (e.g., &lt;!DOCTYPE html PUBLIC &quot;-/W3C//DTD HTML 4.01//EN&quot; &quot;foo&quot;&gt;). In such a case, if the doctype is correctly placed as a document preamble, the parser sets the document to quirks mode.</td>
</tr>
<tr>
<td>absence-of-digits-in-numeric-character-reference</td>
<td>This error occurs if the parser encounters a numeric character reference that doesn't contain any digits (e.g., #qux;). In this case the parser doesn't resolve the character reference.</td>
</tr>
<tr>
<td>cdata-in-html-content</td>
<td>This error occurs if the parser encounters a CDATA section outside of foreign content (SVG or MathML). The parser treats such CDATA sections (including leading &quot;[CDATA[&quot; and trailing &quot;]&quot;) strings) as comments.</td>
</tr>
<tr>
<td>character-reference-outside-unicode-range</td>
<td>This error occurs if the parser encounters a character reference that references a code point that is greater than the valid Unicode range. The parser resolves such a character reference to a U+FFFD REPLACEMENT CHARACTER.</td>
</tr>
<tr>
<td>control-character-in-input-stream</td>
<td>This error occurs if the input stream contains a control code point that is not ASCII whitespace or U+0000 NULL. Such code points are parsed as-is and usually, where parsing rules don't apply any additional restrictions, make their way into the DOM.</td>
</tr>
<tr>
<td>control-character-reference</td>
<td>This error occurs if the parser encounters a numeric character reference that references a control code point that is not ASCII whitespace or is a U+000D CARRIAGE RETURN. The parser resolves such character references as-is except C1 control references that are replaced according to the numeric character reference end state.</td>
</tr>
<tr>
<td>end-tag-with-attributes</td>
<td>This error occurs if the parser encounters an end tag with attributes. Attributes in end tags are completely ignored and do not make their way into the DOM.</td>
</tr>
<tr>
<td>duplicate-attribute</td>
<td>This error occurs if the parser encounters an attribute in a tag that already has an attribute with the same name. The parser ignores all such duplicate occurrences of the attribute.</td>
</tr>
<tr>
<td>end-tag-with-trailing-solidus</td>
<td>This error occurs if the parser encounters an end tag that has a U+002F (/) code point right before the closing U+003E (&gt; ) code point (e.g., »/div/). Such a tag is treated as a regular end tag.</td>
</tr>
<tr>
<td>eol-before-tag-name</td>
<td>This error occurs if the parser encounters the end of the input stream where a tag name is expected. In this case the parser treats the beginning of a start tag (i.e., « ) or an end tag (i.e., » ) as text content.</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>eof-in-cdata</strong></td>
<td>This error occurs if the parser encounters the end of the input stream in a CDATA section. The parser treats such CDATA sections as if they are closed immediately before the end of the input stream.</td>
</tr>
<tr>
<td><strong>eof-in-comment</strong></td>
<td>This error occurs if the parser encounters the end of the input stream in a comment. The parser treats such comments as if they are closed immediately before the end of the input stream.</td>
</tr>
<tr>
<td><strong>eof-in-doctype</strong></td>
<td>This error occurs if the parser encounters the end of the input stream in a DOCTYPE. In such a case, if the DOCTYPE is correctly placed as a document preamble, the parser sets the Document to quirks mode.</td>
</tr>
<tr>
<td><strong>eof-in-script-html-comment-like-text</strong></td>
<td>This error occurs if the parser encounters the end of the input stream in text that resembles an HTML comment inside script elements (e.g., <code>&lt;script&gt;</code>). The parser treats such comments as if they are correctly closed by the <code>--&gt;</code> code point sequence.</td>
</tr>
<tr>
<td><strong>eof-in-tag</strong></td>
<td>This error occurs if the parser encounters the end of the input stream in a start tag or an end tag (e.g., <code>&lt;div id=/&gt;</code>). Such a tag is completely ignored.</td>
</tr>
<tr>
<td><strong>incorrectly-closed-comment</strong></td>
<td>This error occurs if the parser encounters a comment that is closed by the <code>--&gt;</code> code point sequence. The parser treats such comments as if they are correctly closed by the <code>--&gt;</code> code point sequence.</td>
</tr>
<tr>
<td><strong>incorrectly-opened-comment</strong></td>
<td>This error occurs if the parser encounters the <code>&lt;!</code> code point sequence that is not immediately followed by two U+002D (-) code points and that is not the start of a DOCTYPE or a CDATA section. All content that follows the <code>&lt;!</code> code point sequence up to a U+003E (&gt;) code point (if present) or to the end of the input stream is treated as a comment.</td>
</tr>
<tr>
<td><strong>invalid-character-sequence-after-doctype-name</strong></td>
<td>This error occurs if the parser encounters any code point sequence other than &quot;PUBLIC&quot; and &quot;SYSTEM&quot; keywords after a DOCTYPE name. In such a case, the parser ignores any following public or system identifiers, and if the DOCTYPE is correctly placed as a document preamble, sets the Document to quirks mode.</td>
</tr>
<tr>
<td><strong>invalid-first-character-of-tag-name</strong></td>
<td>This error occurs if the parser encounters a code point that is not an ASCII alpha where first code point of a start tag name or an end tag name is expected. If a start tag was expected such code point and a preceding U+003C (&lt;) is treated as text content, and all content that follows is treated as markup. Whereas, if an end tag was expected, such code point and all content that follows up to a U+003E (&gt;) code point (if present) or to the end of the input stream is treated as a comment.</td>
</tr>
<tr>
<td><strong>missing-attribute-value</strong></td>
<td>This error occurs if the parser encounters a U+003E (&gt;) code point where an attribute value is expected (e.g., <code>&lt;div id=/&gt;</code>). The parser treats the attribute as having an empty value.</td>
</tr>
<tr>
<td><strong>missing-doctype-name</strong></td>
<td>This error occurs if the parser encounters a DOCTYPE that is missing a name (e.g., <code>&lt;!DOCTYPE&gt;</code>). In such a case, if the DOCTYPE is correctly placed as a document preamble, the parser sets the Document to quirks mode.</td>
</tr>
<tr>
<td><strong>missing-doctype-public-identifier</strong></td>
<td>This error occurs if the parser encounters a U+003E (&gt;) code point where start of the public identifier is expected (e.g., <code>&lt;!DOCTYPE html PUBLIC &gt;</code>). In such a case, if the DOCTYPE is correctly placed as a document preamble, the parser sets the Document to quirks mode.</td>
</tr>
</tbody>
</table>

**Note**

Syntactic structures that resemble HTML comments in script elements are parsed as text content. They can be a part of a scripting language-specific syntactic structure or be treated as an HTML-like comment, if the scripting language supports them (e.g., parsing rules for HTML-like comments can be found in Annex B of the JavaScript specification). The common reason for this error is a violation of the restrictions for contents of script elements.

**Example**

For example, consider the following markup:

```html
<html><head><body>
#text: <42>
</body></html>
```

This will be parsed into:

```
42
```

**Note**

While the first code point of a tag name is limited to an ASCII alpha, a wide range of code points (including ASCII digits) is allowed in subsequent positions.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>missing-doctype-system-identifier</td>
<td>This error occurs if the parser encounters a U+003E (&gt;) code point where start of the DOCTYPE public identifier is expected (e.g., &lt;!DOCTYPE html SYSTEM&gt;). In such a case, if the DOCTYPE is correctly placed as a document preamble, the parser sets the <code>document</code> to quirks mode.</td>
</tr>
<tr>
<td>missing-end-tag-name</td>
<td>This error occurs if the parser encounters a U+003E (&gt;) code point where an end tag name is expected, i.e., &lt;/&gt;. The parser completely ignores whole &quot;&lt;/&quot; code point sequence.</td>
</tr>
<tr>
<td>missing-quote-before-doctype-public-identifier</td>
<td>This error occurs if the parser encounters the <code>DOCTYPE</code> public identifier that is not preceded by a quote (e.g., &lt;!DOCTYPE html PUBLIC -/W3C//DTD HTML 4.01//EN&gt;). In such a case, the parser ignores the public identifier, and if the DOCTYPE is correctly placed as a document preamble, sets the <code>document</code> to quirks mode.</td>
</tr>
<tr>
<td>missing-quote-before-doctype-system-identifier</td>
<td>This error occurs if the parser encounters the <code>DOCTYPE</code> system identifier that is not preceded by a quote (e.g., &lt;!DOCTYPE html SYSTEM http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd&gt;). In such a case, the parser ignores the system identifier, and if the DOCTYPE is correctly placed as a document preamble, sets the <code>document</code> to quirks mode.</td>
</tr>
<tr>
<td>missing-semicolon-after-character-reference</td>
<td>This error occurs if the parser encounters a character reference that is not terminated by a U+003B (;) code point. Usually the parser behaves as if character reference is terminated by the U+003B (;) code point; however, there are some ambiguous cases in which the parser includes subsequent code points in the character reference.</td>
</tr>
<tr>
<td>missing-whitespace-after-doctype-public-keyword</td>
<td>This error occurs if the parser encounters a <code>DOCTYPE</code> whose &quot;PUBLIC&quot; keyword and public identifier are not separated by ASCII whitespace. In this case the parser behaves as if ASCII whitespace is present.</td>
</tr>
<tr>
<td>missing-whitespace-after-doctype-system-keyword</td>
<td>This error occurs if the parser encounters a <code>DOCTYPE</code> whose &quot;SYSTEM&quot; keyword and system identifier are not separated by ASCII whitespace. In this case the parser behaves as if ASCII whitespace is present.</td>
</tr>
<tr>
<td>missing-whitespace-before-doctype-name</td>
<td>This error occurs if the parser encounters a <code>DOCTYPE</code> whose &quot;DOCTYPE&quot; keyword and name are not separated by ASCII whitespace. In this case the parser behaves as if ASCII whitespace is present.</td>
</tr>
<tr>
<td>missing-whitespace-between-attributes</td>
<td>This error occurs if the parser encounters attributes that are not separated by ASCII whitespace (e.g., &lt;div id=&quot;foo&quot; class=&quot;bar&quot;&gt;). In this case the parser behaves as if ASCII whitespace is present.</td>
</tr>
<tr>
<td>missing-whitespace-between-doctype-public-and-system-identifiers</td>
<td>This error occurs if the parser encounters a <code>DOCTYPE</code> whose public and system identifiers are not separated by ASCII whitespace. In this case the parser behaves as if ASCII whitespace is present.</td>
</tr>
<tr>
<td>nested-comment</td>
<td>This error occurs if the parser encounters a nested comment (e.g., &lt;!-- &lt;!-- nested --&gt; --&gt;). Such a comment will be closed by the first occurring &quot;--&quot; code point sequence and everything that follows will be treated as markup.</td>
</tr>
<tr>
<td>noncharacter-character-reference</td>
<td>This error occurs if the parser encounters a numeric character reference that references a noncharacter. The parser resolves such character references as-is.</td>
</tr>
<tr>
<td>noncharacter-input-stream</td>
<td>This error occurs if the input stream contains a noncharacter. Such code points are parsed as-is and usually, where parsing rules don’t apply any additional restrictions, make their way into the DOM.</td>
</tr>
<tr>
<td>non-void-xml-element-start-tag-with-trailing-solidus</td>
<td>This error occurs if the parser encounters a start tag for an element that is not in the list of void elements or is not a part of foreign content (i.e., not an SVG or MathML element) that has a U+002F (/) code point right before the closing U+003E (&gt;) code point. The parser behaves as if the U+002F (/) is not present.</td>
</tr>
</tbody>
</table>

**Example**

For example, consider the following markup:

```html
<div/><span/></span><span/></span>
```

This will be parsed into:

```
html
head
body
gl
```
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>null-character-reference</td>
<td>This error occurs if the parser encounters a numeric character reference that references a U+0000 NULL code point. The parser resolves such character references to a U+FFFD REPLACEMENT CHARACTER.</td>
</tr>
<tr>
<td>surrogate-character-reference</td>
<td>This error occurs if the parser encounters a numeric character reference that references a surrogate. The parser resolves such character references to a U+FFFD REPLACEMENT CHARACTER.</td>
</tr>
<tr>
<td>surrogate-in-input-stream</td>
<td>This error occurs if the input stream contains a surrogate. Such code points are parsed as-is and usually, where parsing rules don’t apply any additional restrictions, make their way into the DOM.</td>
</tr>
<tr>
<td>unexpected-character-after-doctype-system-identifier</td>
<td>This error occurs if the parser encounters any code points other than ASCII whitespace or closing U+003E (&gt;) after the DOCTYPE system identifier. The parser ignores these code points.</td>
</tr>
<tr>
<td>unexpected-character-in-attribute-name</td>
<td>This error occurs if the parser encounters a U+0022 (&quot;), U+0027 (’), or U+003C (&lt;) code point in an attribute name. The parser includes such code points in the attribute name.</td>
</tr>
<tr>
<td>unexpected-character-in-unquoted-attribute-value</td>
<td>This error occurs if the parser encounters a U+0022 (&quot;), U+0027 (’), U+003C (&lt;), U+003D (=), or U+0060 (`) code point in an unquoted attribute value. The parser includes such code points in the attribute value.</td>
</tr>
</tbody>
</table>

**Example**

For example, consider the following markup:

```html
<div foo="bar"></div>
```

Due to a misplaced U+0027 (’) code point between an attribute name and value the parser treats this markup as a `div` element with the attribute "id'bar" that has an empty value.

**Notes**

- Surrogates can only find their way into the input stream via script APIs such as `document.write()`.
- Code points that trigger this error are usually a part of another syntactic construct and can be a sign of a typo around the attribute name.
- U+0060 (`) is in the list of code points that trigger this error because certain legacy user agents treat it as a quote.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unexpected-equals-sign-before-attribute-name</td>
<td>This error occurs if the parser encounters a U+003D (=) code point before an attribute name. In this case the parser treats U+003D (=) as the first code point of the attribute name.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> The common reason for this error is a forgotten attribute name.</td>
</tr>
<tr>
<td></td>
<td><strong>Example</strong> For example, consider the following markup:</td>
</tr>
<tr>
<td></td>
<td>```xml</td>
</tr>
<tr>
<td></td>
<td>`&lt;div foo=&quot;bar&quot; =&quot;baz&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td>```</td>
</tr>
<tr>
<td></td>
<td>Due to a forgotten attribute name the parser treats this markup as a div element with two attributes: a &quot;foo&quot; attribute with a &quot;bar&quot; value and a &quot;:=&quot; attribute with an empty value.</td>
</tr>
</tbody>
</table>

| unexpected-null-character   | This error occurs if the parser encounters a U+0000 NULL code point in the input stream in certain positions. In general, such code points are either completely ignored or, for security reasons, replaced with a U+FFFD REPLACEMENT CHARACTER. |

| unexpected-question-mark-instead-of-tag-name | This error occurs if the parser encounters a U+003F (?) code point where first code point of a start tag name is expected. The U+003F (?) and all content that follows up to a U+003E (>) code point (if present) or to the end of the input stream is treated as a comment. |
|                                              | **Example** For example, consider the following markup:                                                                                  |
|                                              | ```xml                                                                               |
|                                              | `<?xml-stylesheet type="text/css" href="style.css"?`                                       |
|                                              | ```                                                                                  |
|                                              | This will be parsed into:                                                              |
|                                              | ```html                                                                               |
|                                              | `#comment: xml-stylesheet type="text/css" href="style.css"?`                           |
|                                              | `html`                                                                               |
|                                              | `head`                                                                               |
|                                              | `body`                                                                               |
|                                              | ```                                                                                  |
|                                              | **Note** The common reason for this error is an XML processing instruction (e.g., `<xml-stylesheet type="text/css" href="style.css"?>`) or an XML declaration (e.g., `<?xml version="1.0" encoding="UTF-8"?>`) being used in HTML. |

| unexpected-solidus-in-tag | This error occurs if the parser encounters a U+002F (/) code point that is not a part of a quoted attribute value and not immediately followed by a U+003E (>) code point in a tag (e.g., `<div / id="foo">`). In this case the parser behaves as if it encountered ASCII whitespace. |

| unknown-named-character-reference | This error occurs if the parser encounters an ambiguous ampersand. In this case the parser doesn't resolve the character reference. |

### 13.2.3 The input byte stream

The stream of 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 XML. 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.
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.

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.

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 result of BOM sniffing is an encoding, return that encoding with confidence certain.

   Note: Although the decode algorithm will itself change the encoding to use based on the presence of a byte order mark, this algorithm sniffs the BOM as well in order to set the correct document's character encoding and confidence.

2. 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.

   Note: Typically, user agents remember such user requests across sessions, and in some cases apply them to documents in iframes as well.

3. 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 preparing 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 preparse.
4. If the transport layer specifies a character encoding, and it is supported, return that encoding with the confidence certain.

5. 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 return the same encoding, with confidence tentative.

6. If the HTML parser for which this algorithm is being run is associated with a Document $d$ whose browsing context is non-null and a child browsing context, then:
   1. Let parentDocument be $d$'s browsing context's container document.
   2. If parentDocument's origin is same origin with $d$'s origin and parentDocument's character encoding is not UTF-16BE/LE, then return parentDocument's character encoding, with the confidence tentative.

7. 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.

8. 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.

9. 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.

<table>
<thead>
<tr>
<th>Locale language</th>
<th>Suggested default encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>ar</td>
<td>windows-1256</td>
</tr>
<tr>
<td>ba</td>
<td>windows-1251</td>
</tr>
<tr>
<td>be</td>
<td>windows-1251</td>
</tr>
<tr>
<td>bg</td>
<td>windows-1251</td>
</tr>
<tr>
<td>cs</td>
<td>windows-1250</td>
</tr>
<tr>
<td>el</td>
<td>ISO-8859-7</td>
</tr>
<tr>
<td>et</td>
<td>windows-1257</td>
</tr>
<tr>
<td>fa</td>
<td>windows-1256</td>
</tr>
<tr>
<td>he</td>
<td>windows-1255</td>
</tr>
<tr>
<td>hr</td>
<td>windows-1250</td>
</tr>
</tbody>
</table>

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.

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.
<table>
<thead>
<tr>
<th>Locale language</th>
<th>Suggested default encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>hu</td>
<td>ISO-8859-2</td>
</tr>
<tr>
<td>ja</td>
<td>Shift JIS</td>
</tr>
<tr>
<td>kk</td>
<td>windows-1251</td>
</tr>
<tr>
<td>ko</td>
<td>EUC-KR</td>
</tr>
<tr>
<td>ku</td>
<td>windows-1254</td>
</tr>
<tr>
<td>ky</td>
<td>windows-1251</td>
</tr>
<tr>
<td>lt</td>
<td>windows-1252</td>
</tr>
<tr>
<td>lv</td>
<td>windows-1257</td>
</tr>
<tr>
<td>mk</td>
<td>windows-1251</td>
</tr>
<tr>
<td>pl</td>
<td>ISO-8859-2</td>
</tr>
<tr>
<td>ru</td>
<td>windows-1251</td>
</tr>
<tr>
<td>sah</td>
<td>windows-1251</td>
</tr>
<tr>
<td>sk</td>
<td>windows-1259</td>
</tr>
<tr>
<td>sl</td>
<td>ISO-8859-2</td>
</tr>
<tr>
<td>sr</td>
<td>windows-1251</td>
</tr>
<tr>
<td>tg</td>
<td>windows-1251</td>
</tr>
<tr>
<td>th</td>
<td>windows-874</td>
</tr>
<tr>
<td>tr</td>
<td>windows-1254</td>
</tr>
<tr>
<td>tt</td>
<td>windows-1251</td>
</tr>
<tr>
<td>uk</td>
<td>windows-1251</td>
</tr>
<tr>
<td>vi</td>
<td>windows-1258</td>
</tr>
<tr>
<td>zh-CN</td>
<td>gb18030</td>
</tr>
<tr>
<td>zh-TW</td>
<td>Big5</td>
</tr>
<tr>
<td>All other locales</td>
<td>windows-1252</td>
</tr>
</tbody>
</table>

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 (`<!--`
     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 `<![endif]` 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 sniffed, then jump to the processing step.
If the attribute's name is already in attribute list, then return to the step labeled attributes.

Add the attribute's name to attribute list.

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.

Return to the step labeled attributes.

Processing: If need pragma is null, then jump to the step below labeled next byte.

If need pragma is true but got pragma is false, then jump to the step below labeled next byte.

If charset is failure, then jump to the step below labeled next byte.

If charset is UTF-16BE/LE, then set charset to UTF-8.

If charset is x-user-defined, then set charset to windows-1252.

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 (<), optionally a 0x2F byte (/), and finally a byte in the range 0x41-0x5A or 0x61-0x7A (A-Z or a-z)

1. Advance the position pointer so that it points at the next 0x09 (HT), 0x0A (LF), 0x0C (FF), 0x0D (CR), 0x20 (SP), or 0x3E (>) 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 (`!`)

↓ A sequence of bytes starting with: 0x3C 0x2F (`/`)

↓ A sequence of bytes starting with: 0x3C 0x3F (`?`)

Advance the position pointer so that it points at the first 0x3E byte (>) that comes after the 0x3C byte that was found.

↓ Any other byte
   Do nothing with that byte.

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 (HT), 0x0A (LF), 0x0C (FF), 0x0D (CR), 0x20 (SP), or 0x2F (/) then advance position to the next byte and redo this step.

2. If the byte at position is 0x3E (>) 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 (=), 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 (HT), 0x0A (LF), 0x0C (FF), 0x0D (CR), or 0x20 (SP)
  Jump to the step below labeled spaces.

If it is 0x2F (/) or 0x3E (>)
  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 (A) to 0x5A (Z)
  Append the 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 code point with the same value as the byte at position to attribute name. (It doesn't actually matter how bytes outside the ASCII range are handled here, since only ASCII bytes 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 (HT), 0x0A (LF), 0x0C (FF), 0x0D (CR), or 0x20 (SP) then advance position to the next byte, then, repeat this step.
7. If the byte at position is not 0x3D (=), 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 (=) byte.
9. Value: If the byte at position is one of 0x09 (HT), 0x0A (LF), 0x0C (FF), 0x0D (CR), or 0x20 (SP) then advance position to the next byte, then, repeat this step.
10. Process the byte at position as follows:

   If it is 0x22 (") or 0x27 (')
      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 (A) to 0x5A (Z), then append a code point to attribute value whose value is 0x20 more than the value of the byte at position.
      5. Otherwise, append a code point to attribute value whose value is the same as the value of the byte at position.
      6. Return to the step above labeled quote loop.

   If it is 0x3E (>)
      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 (A) to 0x5A (Z)
      Append a 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 a code point with the same value as 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 (HT), 0x0A (LF), 0x0C (FF), 0x0D (CR), 0x20 (SP), or 0x3E (>)
      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 (A) to 0x5A (Z)
      Append a code point b+0x20 to attribute value (where b is the value of the byte at position).
Append a code point with the same value as 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.)

13.2.3.3 Character 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. 

13.2.3.4 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 UTF-16BE/LE, then set the confidence to certain and return. 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 UTF-16BE/LE, then change it to UTF-8.

3. If the new encoding is x-user-defined, then change it to windows-1252.

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 return. 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 return.

6. Otherwise, navigate to the document again, with historyHandling set to "replace", 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.

13.2.3.5 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 surrogates are surrogate-in-input-stream parse errors. Any occurrences of noncharacters are noncharacter-in-input-stream parse errors, and any occurrences of controls other than ASCII whitespace and U+0000 NULL characters are control-character-in-input-stream parse errors.

Note
The handling of U+0000 NULL characters varies based on where the characters are found and happens at the later stages of the parsing. They are either ignored or, for security reasons, replaced with a U+FFFD REPLACEMENT CHARACTER. This handling is, by necessity, spread across both the tokenization stage and the tree construction stage.

Before the tokenization stage, the input stream must be preprocessed by normalizing newlines. Thus, newlines in HTML DOMs are represented by U+000A LF characters, and there are never any U+000D 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.

13.2.4 Parse state

13.2.4.1 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", "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 UA to reset the insertion mode appropriately, it means the UA 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 created as part of
the HTML fragment parsing algorithm, 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" and return.
   7. Jump back to the step labeled loop.
   8. Done: Switch the insertion mode to "in select" and return.

5. If node is a td or th element and last is false, then switch the insertion mode to "in cell" and return.

8. If node is a tr element, then switch the insertion mode to "in row" and return.

11. If node is a thead, tbody, or tfoot element, then switch the insertion mode to "in table" and return.

14. If node is a frameset element, then switch the insertion mode to "in frameset" and return. (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 return. (fragment case)
   2. Otherwise, the head element pointer is not null, switch the insertion mode to "after head" and return.

16. If last is true, then switch the insertion mode to "in body" and return. (fragment case)

17. Let node now be the node before node in the stack of open elements.

18. Return to the step labeled loop.

13.2.4.2 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 document element, 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 finishes.

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 as part of 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


#### Note

An image start tag token is handled by the tree builder, but it is not in this list because it is not an element; it gets turned into an **img** element.

### Formatting

The following HTML elements are those that end up in the **list of active formatting elements**: **a**, **b**, **big**, **code**, **em**, **font**, **i**, **nobr**, **s**, **small**, **strong**, **tt**, and **u**.

### Ordinary

All other elements found while parsing an HTML document.

#### Note

Typically, the **special** elements have the start and end tags handled specifically, while **ordinary** elements' tokens fall into "any other start tag" and "any other end tag" clauses, and some parts of the tree builder check if a particular element in the **stack of open elements** is in the **special** category. However, some elements (e.g., the **option** element) have their start or end tag tokens handled specifically, but are still not in the **special** category, so that they get the **ordinary** handling elsewhere.

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**, **caption**,

1086
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.
- ol in the HTML namespace
- ul 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.

### 13.2.4.3 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, object, marquee, template, td, th, and caption elements, and are used to prevent formatting from "leaking" into applet, object, marquee, template, td, th, and caption elements.

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 UA to push onto the list of active formatting elements an element element, the UA 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).
2. Add element to the list of active formatting elements.

When the steps below require the UA to reconstruct the active formatting elements, the UA 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.

When the steps below require the UA to clear the list of active formatting elements up to the last marker, the UA 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.

13.2.4.4 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.

13.2.4.5 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.
The frameset-ok flag is set to "ok" when the parser is created. It is set to "not ok" after certain tokens are seen.

13.2.5 Tokenization

Implementations must act as if they used the following state machine to tokenize HTML. The state machine must start in the 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 current input 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.

When a state says to reconsume a matched character in a specified state, that means to switch to that state, but when it attempts to consume the next input character, provide it with the current input character instead.

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, and the character reference state uses a return state to return to the state it was invoked from.

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, 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.)

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 non-void-html-element-start-tag-with-trailing-solidus parse error.

When an end tag token is emitted with attributes, that is an end-tag-with-attributes parse error.

When an end tag token is emitted with its self-closing flag set, that is an end-tag-with-trailing-solidus 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.

A character reference is said to be consumed as part of an attribute if the return state is either attribute value (double-quoted) state or attribute value (single-quoted) state or attribute value (unquoted) state.

When a state says to flush code points consumed as a character reference, it means that for each code point in the temporary buffer (in the order they were added to the buffer) user agent must append the code point from the buffer to the current attribute's value if the character reference was consumed as part of an attribute, or emit the code point as a character token otherwise.

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.
13.2.5.1 **Data state**

Consume the next input character.

* U+0026 AMPERSAND (&)
  
  Set the return state to the *data state*. Switch to the *character reference state*.

* U+003C LESS-THAN SIGN (<)
  
  Switch to the *tag open state*.

* U+0000 NULL
  
  This is an unexpected-null-character 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.

13.2.5.2 **RCDATA state**

Consume the next input character:

* U+0026 AMPERSAND (&)
  
  Set the return state to the *RCDATA state*. Switch to the *character reference state*.

* U+003C LESS-THAN SIGN (<)
  
  Switch to the *RCDATA less-than sign state*.

* U+0000 NULL
  
  This is an unexpected-null-character 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.

13.2.5.3 **RAWTEXT state**

Consume the next input character:

* U+003C LESS-THAN SIGN (<)
  
  Switch to the *RAWTEXT less-than sign state*.

* U+0000 NULL
  
  This is an unexpected-null-character 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.

13.2.5.4 **Script data state**

Consume the next input character:

* U+003C LESS-THAN SIGN (<)
  
  Switch to the *script data less-than sign state*.
This is an unexpected-null-character parse error. Emit a U+FFFD REPLACEMENT CHARACTER character token.

Emit an end-of-file token.

Emit the current input character as a character token.

13.2.5.5 PLAINTEXT state

Consume the next input character:

This is an unexpected-null-character parse error. Emit a U+FFFD REPLACEMENT CHARACTER character token.

Emit an end-of-file token.

Emit the current input character as a character token.

13.2.5.6 Tag open state

Consume the next input character:

Switch to the markup declaration open state.

Switch to the end tag open state.

Create a new start tag token, set its tag name to the empty string. Reconsume in the tag name state.

This is an unexpected-question-mark-instead-of-tag-name parse error. Create a comment token whose data is the empty string. Reconsume in the bogus comment state.

This is an eof-before-tag-name parse error. Emit a U+003C LESS-THAN SIGN character token and an end-of-file token.

This is an invalid-first-character-of-tag-name parse error. Emit a U+003C LESS-THAN SIGN character token.

Reconsume in the data state.

13.2.5.7 End tag open state

Consume the next input character:

Create a new end tag token, set its tag name to the empty string. Reconsume in the tag name state.

This is a missing-end-tag-name parse error. Switch to the data state.

This is an eof-before-tag-name parse error. Emit a U+003C LESS-THAN SIGN character token, a U+002F SOLIDUS character token and an end-of-file token.
Anything else
This is an invalid-first-character-of-tag-name parse error. Create a comment token whose data is the empty string. Reconsume in the bogus comment state.

13.2.5.8 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 before attribute name state.
- U+002F SOLIDUS (/)
  Switch to the self-closing start tag state.
- U+003E GREATER-THAN SIGN (>)
  Switch to the data state. Emit the current tag token.

- ASCII upper alpha
  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
  This is an unexpected-null-character parse error. Append a U+FFFD REPLACEMENT CHARACTER character to the current tag token's tag name.

- EOF
  This is an eof-in-tag parse error. Emit an end-of-file token.

- Anything else
  Append the current input character to the current tag token's tag name.

13.2.5.9 RCDATA less-than sign state
Consume the next input character:

- U+002F SOLIDUS (/)
  Set the temporary buffer to the empty string. Switch to the RCDATA end tag open state.

- Anything else
  Emit a U+003C LESS-THAN SIGN character token. Reconsume in the RCDATA state.

13.2.5.10 RCDATA end tag open state
Consume the next input character:

- ASCII alpha
  Create a new end tag token, set its tag name to the empty string. Reconsume in the RCDATA end tag name state.

- Anything else
  Emit a U+003C LESS-THAN SIGN character token and a U+002F SOLIDUS character token. Reconsume in the RCDATA state.
13.2.5.11 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 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 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 data state and emit the current tag token. Otherwise, treat it as per the "anything else" entry below.
- ASCII upper alpha
  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.
- ASCII lower alpha
  Append the current input character to the current tag token's tag name. Append the current input character to the temporary buffer.
- Anything else
  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 in the RCDATA state.

13.2.5.12 RAWTEXT less-than sign state

Consume the next input character:

- U+002F SOLIDUS (/)
  Set the temporary buffer to the empty string. Switch to the RAWTEXT end tag open state.
- Anything else
  Emit a U+003C LESS-THAN SIGN character token. Reconsume in the RAWTEXT state.

13.2.5.13 RAWTEXT end tag open state

Consume the next input character:

- ASCII alpha
  Create a new end tag token, set its tag name to the empty string. Reconsume in the RAWTEXT end tag name state.
- Anything else
  Emit a U+003C LESS-THAN SIGN character token and a U+002F SOLIDUS character token. Reconsume in the RAWTEXT state.

13.2.5.14 RAWTEXT end tag name state

Consume the next input character:
If the current end tag token is an **appropriate end tag token**\(^{1089}\), then switch to the **before attribute name state**\(^{1100}\). Otherwise, treat it as per the "anything else" entry below.

**U+0020 SPACE**

If the current end tag token is an **appropriate end tag token**\(^{1089}\), then switch to the **self-closing start tag state**\(^{1103}\). 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**\(^{1089}\), then switch to the **data state**\(^{1090}\) and emit the current tag token. Otherwise, treat it as per the "anything else" entry below.

**ASCII upper alpha**

Append the lowercase version of the current input character\(^{1084}\) (add 0x0020 to the character’s code point) to the current tag token's tag name. Append the current input character\(^{1084}\) to the temporary buffer\(^{1089}\).

**ASCII lower alpha**

Append the current input character\(^{1084}\) to the current tag token's tag name. Append the current input character\(^{1084}\) to the temporary buffer\(^{1089}\).

**Anything else**

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\(^{1089}\) (in the order they were added to the buffer). **Reconsume**\(^{1089}\) in the **RAWTEXT state**\(^{1090}\).

### 13.2.5.15 Script data less-than sign state \(^{5\text{p10}\text{94}}\)

Consume the next input character\(^{1084}\):

**U+002F SOLIDUS (/)**

Set the temporary buffer\(^{1089}\) to the empty string. Switch to the **script data end tag open state**\(^{1094}\).

**U+0021 EXCLAMATION MARK (!)**

Switch to the **script data escape start state**\(^{1095}\). Emit a U+003C LESS-THAN SIGN character token and a U+0021 EXCLAMATION MARK character token.

**Anything else**

Emit a U+003C LESS-THAN SIGN character token. **Reconsume**\(^{1089}\) in the **script data state**\(^{1090}\).

### 13.2.5.16 Script data end tag open state \(^{5\text{p10}\text{94}}\)

Consume the next input character\(^{1084}\):

**ASCII alpha**

Create a new end tag token, set its tag name to the empty string. **Reconsume**\(^{1089}\) in the **script data end tag name state**\(^{1094}\).

**Anything else**

Emit a U+003C LESS-THAN SIGN character token and a U+002F SOLIDUS character token. **Reconsume**\(^{1089}\) in the **script data state**\(^{1090}\).

### 13.2.5.17 Script data end tag name state \(^{5\text{p10}\text{94}}\)

Consume the next input character\(^{1084}\):

1094
13.2.5.18 Script data escape start state

Consume the next input character:

- U+002D HYPHEN-MINUS (-)
  Switch to the script data escape start dash state. Emit a U+002D HYPHEN-MINUS character token.

- Anything else
  Reconsume in the script data state.

13.2.5.19 Script data escape start dash state

Consume the next input character:

- U+002D HYPHEN-MINUS (-)
  Switch to the script data escaped dash dash state. Emit a U+002D HYPHEN-MINUS character token.

- Anything else
  Reconsume in the script data state.

13.2.5.20 Script data escaped state

Consume the next input character:

- U+002D HYPHEN-MINUS (-)
  Switch to the script data escaped dash dash state. Emit a U+002D HYPHEN-MINUS character token.

- U+003C LESS-THAN SIGN (<)
  Switch to the script data escaped less-than sign state.

- U+0000 NULL
  This is an unexpected-null-character parse error. Emit a U+FFFD REPLACEMENT CHARACTER character token.
This is an `eof-in-script-html-comment-like-text` parse error. Emit an end-of-file token.

Anything else

Emit the `current input character` as a character token.

### 13.2.5.21 Script data escaped dash state

Consume the `next input character`:

* U+002D HYPHEN-MINUS (-)
  
Switch to the `script data escaped dash dash state`. Emit a U+002D HYPHEN-MINUS character token.

* U+003C LESS-THAN SIGN (<)
  
Switch to the `script data escaped less-than sign state`.

* U+0000 NULL
  
This is an `unexpected-null-character` parse error. Switch to the `script data escaped state`. Emit a U+FFFD REPLACEMENT CHARACTER character token.

* EOF
  
This is an `eof-in-script-html-comment-like-text` parse error. Emit an end-of-file token.

Anything else

Switch to the `script data escaped state`. Emit the `current input character` as a character token.

### 13.2.5.22 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 `script data escaped less-than sign state`.

* U+003E GREATER-THAN SIGN (>)
  
Switch to the `script data state`. Emit a U+003E GREATER-THAN SIGN character token.

* U+0000 NULL
  
This is an `unexpected-null-character` parse error. Switch to the `script data escaped state`. Emit a U+FFFD REPLACEMENT CHARACTER character token.

* EOF
  
This is an `eof-in-script-html-comment-like-text` parse error. Emit an end-of-file token.

Anything else

Switch to the `script data escaped state`. Emit the `current input character` as a character token.

### 13.2.5.23 Script data escaped less-than sign state

Consume the `next input character`:

* U+007F SOLIDUS (/)
  
Set the `temporary buffer` to the empty string. Switch to the `script data escaped end tag open state`.

* ASCII alpha
  
Set the `temporary buffer` to the empty string. Emit a U+003C LESS-THAN SIGN character token. **Reconsume** in the `script data double escape start state`.
Anything else

Emit a U+003C LESS-THAN SIGN character token. \texttt{Reconsume} in the \texttt{script data escaped state}.

### 13.2.5.24 Script data escaped end tag open state

Consume the \texttt{next input character}:

- \texttt{ASCII alpha}

  Create a new end tag token, set its tag name to the empty string. \texttt{Reconsume} in the \texttt{script data escaped end tag name state}.

- Anything else

  Emit a U+003C LESS-THAN SIGN character token and a U+002F SOLIDUS character token. \texttt{Reconsume} in the \texttt{script data escaped state}.

### 13.2.5.25 Script data escaped end tag name state

Consume the \texttt{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 \texttt{appropriate end tag token}, then switch to the \texttt{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 \texttt{appropriate end tag token}, then switch to the \texttt{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 \texttt{appropriate end tag token}, then switch to the \texttt{data state} and emit the current tag token. Otherwise, treat it as per the "anything else" entry below.

- \texttt{ASCII upper alpha}

  Append the lowercase version of the \texttt{current input character} (add 0x0020 to the character's code point) to the current tag token's tag name. Append the \texttt{current input character} to the \texttt{temporary buffer}.

- \texttt{ASCII lower alpha}

  Append the \texttt{current input character} to the current tag token's tag name. Append the \texttt{current input character} to the \texttt{temporary buffer}.

- Anything else

  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 \texttt{temporary buffer} (in the order they were added to the buffer). \texttt{Reconsume} in the \texttt{script data escaped state}.

### 13.2.5.26 Script data double escape start state

Consume the \texttt{next input character}:
If the temporary buffer is the string "script", then switch to the script data double escaped state. Otherwise, switch to the script data escaped state. Emit the current input character as a character token.

ASCII upper alpha

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.

ASCII lower alpha

Append the current input character to the temporary buffer. Emit the current input character as a character token.

Anything else

Reconsume in the script data escaped state.

13.2.5.27 Script data double escaped state

Consume the next input character:

U+002D HYPHEN-MINUS (-)

Switch to the script data double escaped dash state. Emit a U+002D HYPHEN-MINUS character token.

U+003C LESS-THAN SIGN (<)

Switch to the script data double escaped less-than sign state. Emit a U+003C LESS-THAN SIGN character token.

U+0000 NULL

This is an unexpected-null-character parse error. Emit a U+FFFD REPLACEMENT CHARACTER character token.

EOF

This is an eof-in-script-html-comment-like-text parse error. Emit an end-of-file token.

Anything else

Switch to the script data double escaped state. Emit the current input character as a character token.

13.2.5.28 Script data double escaped dash state

Consume the next input character:

U+002D HYPHEN-MINUS (-)

Switch to the script data double escaped dash dash state. Emit a U+002D HYPHEN-MINUS character token.

U+003C LESS-THAN SIGN (<)

Switch to the script data double escaped less-than sign state. Emit a U+003C LESS-THAN SIGN character token.

U+0000 NULL

This is an unexpected-null-character parse error. Switch to the script data double escaped state. Emit a U+FFFD REPLACEMENT CHARACTER character token.

EOF

This is an eof-in-script-html-comment-like-text parse error. Emit an end-of-file token.

Anything else

Switch to the script data double escaped state. Emit the current input character as a character token.
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 script data double escaped less-than sign state[^1099]. Emit a U+003C LESS-THAN SIGN character token.

- **U+003E GREATER-THAN SIGN (>)**
  Switch to the script data state[^1099]. Emit a U+003E GREATER-THAN SIGN character token.

- **U+0000 NULL**
  This is an unexpected-null-character[^1077] parse error[^1073]. Switch to the script data double escaped state[^1098]. Emit a U+FFFD REPLACEMENT CHARACTER character token.

- **EOF**
  This is an eof-in-script-html-comment-like-text[^1074] parse error[^1073]. Emit an end-of-file token.

- **Anything else**
  Switch to the script data double escaped state[^1098]. Emit the current input character[^1084] as a character token.

### 13.2.5.30 Script data double escaped less-than sign state[^1099]

Consume the next input character[^1084]:

- **U+002F SOLIDUS (/)**
  Set the temporary buffer[^1089] to the empty string. Switch to the script data double escape end state[^1099]. Emit a U+002F SOLIDUS character token.

- **Anything else**
  Recomsume[^1089] in the script data double escaped state[^1098].

### 13.2.5.31 Script data double escape end state[^1099]

Consume the next input character[^1084]:

- **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[^1089] is the string "script", then switch to the script data escaped state[^1095]. Otherwise, switch to the script data double escaped state[^1098]. Emit the current input character[^1084] as a character token.

- **ASCII upper alpha**
  Append the lowercase version of the current input character[^1084] (add 0x0020 to the character's code point) to the temporary buffer[^1089]. Emit the current input character[^1084] as a character token.

- **ASCII lower alpha**
  Append the current input character[^1084] to the temporary buffer[^1089]. Emit the current input character[^1084] as a character token.

- **Anything else**
  Recomsume[^1089] in the script data double escaped state[^1098].
13.2.5.32 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 (/)
- U+003E GREATER-THEAN SIGN (>)
- EOF
  - Reconsume in the after attribute name state.

- U+003D EQUALS SIGN (=)
  - This is an unexpected-equals-sign-before-attribute-name parse error. 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 attribute name state.
  - Anything else
    - Start a new attribute in the current tag token. Set that attribute name and value to the empty string. Reconsume in the attribute name state.

13.2.5.33 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
- U+002F SOLIDUS (/)
- U+003E GREATER-THEAN SIGN (>)
- EOF
  - Reconsume in the after attribute name state.

- U+003D EQUALS SIGN (=)
  - Switch to the before attribute value state.

- ASCII upper alpha
  - 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
  - This is an unexpected-null-character 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-THEAN SIGN (<)
  - 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 duplicate-attribute parse error and the new attribute must be removed from the token.
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.

13.2.5.34 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 self-closing start tag state.
- U+003D EQUALS SIGN (=)
  - Switch to the before attribute value state.
- U+003E GREATER-THAN SIGN (>
  - Switch to the data state. Emit the current tag token.
- EOF
  - This is an eof-in-tag parse error. Emit an end-of-file token.
- Anything else
  - Start a new attribute in the current tag token. Set that attribute name and value to the empty string. Recompose in the attribute name state.

13.2.5.35 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 attribute value (double-quoted) state.
- U+0027 APOSTROPHE (’)
  - Switch to the attribute value (single-quoted) state.
- U+003E GREATER-THAN SIGN (>
  - This is a missing-attribute-value parse error. Switch to the data state. Emit the current tag token.
- Anything else
  - Recompose in the attribute value (unquoted) state.

13.2.5.36 Attribute value (double-quoted) state

Consume the next input character:
13.2.5.37 Attribute value (single-quoted) state

Consume the next input character:

- U+0027 APOSTROPHE (')
  Switch to the after attribute value (quoted) state.
- U+0026 AMPERSAND (&)
  Set the return state to the attribute value (single-quoted) state. Switch to the character reference state.
- U+0000 NULL
  This is an unexpected-null-character parse error. Append a U+FFFD REPLACEMENT CHARACTER character to the current attribute's value.
- EOF
  This is an eof-in-tag parse error. Emit an end-of-file token.
- Anything else
  Append the current input character to the current attribute's value.

13.2.5.38 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 before attribute name state.
- U+0026 AMPERSAND (&)
  Set the return state to the attribute value (unquoted) state. Switch to the character reference state.
- U+003E GREATER-THAN SIGN (>)
  Switch to the data state. Emit the current tag token.
- U+0000 NULL
  This is an unexpected-null-character parse error. Append a U+FFFD REPLACEMENT CHARACTER character to the current attribute's value.
13.2.5.39 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 before attribute name state.
→ U+002F SOLIDUS (/)
     Switch to the self-closing start tag state.
→ U+003E GREATER-THAN SIGN (>)
     Switch to the data state. Emit the current tag token.
→ EOF
     This is an eof-in-tag, parse error. Emit an end-of-file token.
→ Anything else
     This is a missing-whitespace-between-attributes parse error. Reconsume in the before attribute name state.

13.2.5.40 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 data state. Emit the current tag token.
→ EOF
     This is an eof-in-tag, parse error. Emit an end-of-file token.
→ Anything else
     This is an unexpected-solidus-in-tag parse error. Reconsume in the before attribute name state.

13.2.5.41 Bogus comment state

Consume the next input character:

→ U+003E GREATER-THAN SIGN (>)
     Switch to the data state. Emit the comment token.
→ EOF
     Emit the comment. Emit an end-of-file token.
This is an unexpected-null-character parse error. Append a U+FFFD REPLACEMENT CHARACTER character to the comment token's data.

Anything else
Append the current input character to the comment token's data.

13.2.5.42 Markup declaration open state
If the next few characters are:

Two U+002D HYPHEN-MINUS characters (-)
Consume those two characters, create a comment token whose data is the empty string, and switch to the comment start state.

ASCII case-insensitive match for the word "DOCTYPE"
Consume those characters and switch to the DOCTYPE state.

The string "[CDATA[" (the five uppercase letters "CDATA" with a U+005B LEFT SQUARE BRACKET character before and after)
Consume those characters. If there is an adjusted current node and it is not an element in the HTML namespace, then switch to the CDATA section state. Otherwise, this is a cdata-in-html-content parse error. Create a comment token whose data is the "[CDATA[" string. Switch to the bogus comment state.

Anything else
This is an incorrectly-opened-comment parse error. Create a comment token whose data is the empty string. Switch to the bogus comment state (don't consume anything in the current state).

13.2.5.43 Comment start state
Consume the next input character:

U+002D HYPHEN-MINUS (-)
Switch to the comment start dash state.

U+003E GREATER-THEAN SIGN (>)
This is an abrupt-closing-of-empty-comment parse error. Switch to the data state. Emit the comment token.

Anything else
Reconsume in the comment state.

13.2.5.44 Comment start dash state
Consume the next input character:

U+002D HYPHEN-MINUS (-)
Switch to the comment end state.

U+003E GREATER-THEAN SIGN (>)
This is an abrupt-closing-of-empty-comment parse error. Switch to the data state. Emit the comment token.

EOF
This is an eof-in-comment parse error. Emit the comment token. Emit an end-of-file token.

Anything else
Append a U+002D HYPHEN-MINUS character (-) to the comment token's data. Reconsume in the comment state.
13.2.5.45 Comment state 

Consume the next input character.

↪ U+003C LESS-THAN SIGN (<)
   Append the current input character to the comment token's data. Switch to the comment less-than sign state.

↪ U+002D HYPHEN-MINUS (-)
   Switch to the comment less-than sign dash state.

↪ U+0000 NULL
   This is an unexpected-null-character parse error. Append a U+FFFD REPLACEMENT CHARACTER character to the comment token's data.

↪ EOF
   This is an eof-in-comment parse error. Emit the comment token. Emit an end-of-file token.

↪ Anything else
   Append the current input character to the comment token's data.

13.2.5.46 Comment less-than sign state

Consume the next input character:

↪ U+0021 EXCLAMATION MARK (!)
   Append the current input character to the comment token's data. Switch to the comment less-than sign bang state.

↪ U+003C LESS-THAN SIGN (<)
   Append the current input character to the comment token's data.

↪ Anything else
   Reconsume in the comment state.

13.2.5.47 Comment less-than sign bang state

Consume the next input character:

↪ U+002D HYPHEN-MINUS (-)
   Switch to the comment less-than sign bang dash state.

↪ Anything else
   Reconsume in the comment state.

13.2.5.48 Comment less-than sign bang dash state

Consume the next input character:

↪ U+002D HYPHEN-MINUS (-)
   Switch to the comment less-than sign bang dash dash state.

↪ Anything else
   Reconsume in the comment end dash state.

13.2.5.49 Comment less-than sign bang dash dash state

Consume the next input character.
13.2.5.50 Comment end dash state

Consume the next input character:

- U+003E GREATER-THAN SIGN (>)
  Switch to the comment end state.
- EOF
  This is a nested-comment parse error. Reconsume in the comment end state.
- Anything else
  Append a U+002D HYPHEN-MINUS character (-) to the comment token's data. Reconsume in the comment state.

13.2.5.51 Comment end state

Consume the next input character:

- U+003E GREATER-THAN SIGN (>)
  Switch to the data state. Emit the comment token.
- U+0021 EXCLAMATION MARK (!)
  Switch to the comment end bang state.
- U+002D HYPHEN-MINUS (-)
  Append a U+002D HYPHEN-MINUS character (-) to the comment token's data.
- EOF
  This is an eof-in-comment parse error. Emit the comment token. Emit an end-of-file token.
- Anything else
  Append two U+002D HYPHEN-MINUS characters (-) to the comment token's data. Reconsume in the comment state.

13.2.5.52 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 comment end dash state.
- U+003E GREATER-THAN SIGN (>)
  This is an incorrectly-closed-comment parse error. Switch to the data state. Emit the comment token.
- EOF
  This is an eof-in-comment parse error. Emit the comment token. Emit an end-of-file token.
- Anything else
  Append two U+002D HYPHEN-MINUS characters (-) and a U+0021 EXCLAMATION MARK character (!) to the comment token's data. Reconsume in the comment state.
13.2.5.53 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 before DOCTYPE name state.

- U+003E GREATER-THAN SIGN (>)

  Recomsume in the before DOCTYPE name state.

- EOF

  This is an invalid parse error. Create a new DOCTYPE token. Set its force-quirks flag to on. Emit the token. Emit an end-of-file token.

- Anything else

  This is a missing whitespace before DOCTYPE name parse error. Recomsume in the before DOCTYPE name state.

13.2.5.54 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.

- ASCII upper alpha

  Create a new DOCTYPE token. Set the token's name to the lowercase version of the current input character (add 0x020 to the character's code point). Switch to the DOCTYPE name state.

- U+0000 NULL

  This is an unexpected null character parse error. Create a new DOCTYPE token. Set the token's name to a U+FFFD replacement character. Switch to the DOCTYPE name state.

- U+003E GREATER-THAN SIGN (>)

  This is a missing doctype name parse error. Create a new DOCTYPE token. Set its force-quirks flag to on. Switch to the data state. Emit the token.

- EOF

  This is an invalid parse error. Create a new DOCTYPE token. Set its force-quirks flag to on. Emit the token. Emit an end-of-file token.

- Anything else

  Create a new DOCTYPE token. Set the token's name to the current input character. Switch to the DOCTYPE name state.

13.2.5.55 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 after DOCTYPE name state.
13.2.5.56 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 data state. Emit the current DOCTYPE token.

- ASCII upper alpha
  
  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
  
  This is an unexpected-null-character parse error. Append a U+FFFD REPLACEMENT CHARACTER character to the current DOCTYPE token's name.

- EOF
  
  This is an eof-in-doctype parse error. Set the DOCTYPE token's force-quirks flag to on. Emit that DOCTYPE token.
  
  Emit an end-of-file token.

- Anything else
  
  Append the current input character to the current DOCTYPE token's name.

13.2.5.57 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 before DOCTYPE public identifier state.

- U+0022 QUOTATION MARK ("")
  
  This is a missing-whitespace-after-doctype-public-keyword parse error. Set the DOCTYPE token's public identifier to the empty string (not missing), then switch to the DOCTYPE public identifier (double-quoted) state.

- U+0027 APOSTROPHE (‘)
  
  This is a missing-whitespace-after-doctype-public-keyword parse error. Set the DOCTYPE token's public identifier to the
empty string (not missing), then switch to the **DOCTYPE public identifier (single-quoted) state**.

- **U+003E GREATER-THEAN SIGN (>)**
  - This is a *missing-doctype-public-identifier* *parse error*. Set the DOCTYPE token's *force-quirks flag* to on. Switch to the **data state**. Emit that DOCTYPE token.

- **EOF**
  - This is an *eof-in-doctype* *parse error*. Set the DOCTYPE token's *force-quirks flag* to on. Emit that DOCTYPE token. Emit an end-of-file token.

- **Anything else**
  - This is a *missing-quote-before-doctype-public-identifier* *parse error*. Set the DOCTYPE token's *force-quirks flag* to on. Recomsume in the **bogus DOCTYPE state**.

### 13.2.5.58 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 **DOCTYPE public identifier (double-quoted) state**.
- **U+0027 APOSTROPH (')**
  - Set the DOCTYPE token's public identifier to the empty string (not missing), then switch to the **DOCTYPE public identifier (single-quoted) state**.
- **U+003E GREATER-THEAN SIGN (>)**
  - This is a *missing-doctype-public-identifier* *parse error*. Set the DOCTYPE token's *force-quirks flag* to on. Switch to the **data state**. Emit that DOCTYPE token.
- **EOF**
  - This is an *eof-in-doctype* *parse error*. Set the DOCTYPE token's *force-quirks flag* to on. Emit that DOCTYPE token. Emit an end-of-file token.
- **Anything else**
  - This is a *missing-quote-before-doctype-public-identifier* *parse error*. Set the DOCTYPE token's *force-quirks flag* to on. Recomsume in the **bogus DOCTYPE state**.

### 13.2.5.59 DOCTYPE public identifier (double-quoted) state

Consume the **next input character**:

- **U+0022 QUOTATION MARK (")**
  - Switch to the **after DOCTYPE public identifier state**.
- **U+0000 NULL**
  - This is an *unexpected-null-character* *parse error*. Append a U+FFFD REPLACEMENT CHARACTER character to the current DOCTYPE token's public identifier.
- **U+003E GREATER-THEAN SIGN (>)**
  - This is an *abrupt-doctype-public-identifier* *parse error*. Set the DOCTYPE token's *force-quirks flag* to on. Switch to the **data state**. Emit that DOCTYPE token.
- **EOF**
  - This is an *eof-in-doctype* *parse error*. Set the DOCTYPE token's *force-quirks flag* to on. Emit that DOCTYPE token. Emit an end-of-file token.
Anything else
Append the current input character to the current DOCTYPE token's public identifier.

13.2.5.60 DOCTYPE public identifier (single-quoted) state

Consume the next input character:

- U+0027 APOSTROPHE (')
  Switch to the after DOCTYPE public identifier state.
- U+0000 NULL
  This is an unexpected-null-character parse error. Append a U+FFFD REPLACEMENT CHARACTER character to the current DOCTYPE token's public identifier.
- U+003E GREATER-THAN SIGN (>)
  This is an abrupt-doctype-public-identifier parse error. Set the DOCTYPE token's force-quirks flag to on. Switch to the data state. Emit that DOCTYPE token.
- EOF
  This is an eof-in-doctype parse error. Set the DOCTYPE token's force-quirks flag to on. Emit that DOCTYPE token. Emit an end-of-file token.
- Anything else
  Append the current input character to the current DOCTYPE token's public identifier.

13.2.5.61 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 between DOCTYPE public and system identifiers state.
- U+003E GREATER-THAN SIGN (>)
  Switch to the data state. Emit the current DOCTYPE token.
- U+0022 QUOTATION MARK ("")
  This is a missing-whitespace-between-doctype-public-and-system-identifiers parse error. Set the DOCTYPE token's system identifier to the empty string (not missing), then switch to the DOCTYPE system identifier (double-quoted) state.
- U+0027 APOSTROPHE (')
  This is a missing-whitespace-between-doctype-public-and-system-identifiers parse error. Set the DOCTYPE token's system identifier to the empty string (not missing), then switch to the DOCTYPE system identifier (single-quoted) state.
- EOF
  This is an eof-in-doctype parse error. Set the DOCTYPE token's force-quirks flag to on. Emit that DOCTYPE token. Emit an end-of-file token.
- Anything else
  This is a missing-quote-before-doctype-system-identifier parse error. Set the DOCTYPE token's force-quirks flag to on. Reconsume in the bogus DOCTYPE state.

13.2.5.62 Between DOCTYPE public and system identifiers state

Consume the next input character:
13.2.5.63 After DOCTYPE system keyword state §11

13.2.5.64 Before DOCTYPE system identifier state §11
Set the DOCTYPE token's system identifier to the empty string (not missing), then switch to the **DOCTYPE system identifier (double-quoted) state**.

Set the DOCTYPE token's system identifier to the empty string (not missing), then switch to the **DOCTYPE system identifier (single-quoted) state**.

This is a **missing-doctype-system-identifier** parse error. Set the DOCTYPE token's **force-quirks flag** to on. Switch to the **data state**. Emit that DOCTYPE token.

This is an **eof-in-doctype** parse error. Set the DOCTYPE token's **force-quirks flag** to on. Emit that DOCTYPE token.

Emit an end-of-file token.

This is a **missing-quote-before-doctype-system-identifier** parse error. Set the DOCTYPE token's **force-quirks flag** to on. Reconsume in the **bogus DOCTYPE state**.

Consume the next input character:

Switch to the **after DOCTYPE system identifier state**.

This is an **unexpected-null-character** parse error. Append a U+FFFD REPLACEMENT CHARACTER character to the current DOCTYPE token's system identifier.

This is an **abrupt-doctype-system-identifier** parse error. Set the DOCTYPE token's **force-quirks flag** to on. Switch to the **data state**. Emit that DOCTYPE token.

This is an **eof-in-doctype** parse error. Set the DOCTYPE token's **force-quirks flag** to on. Emit that DOCTYPE token. Emit an end-of-file token.

Append the current input character to the current DOCTYPE token's system identifier.

Consume the next input character:

Switch to the **after DOCTYPE system identifier state**.

This is an **unexpected-null-character** parse error. Append a U+FFFD REPLACEMENT CHARACTER character to the current DOCTYPE token's system identifier.

This is an **abrupt-doctype-system-identifier** parse error. Set the DOCTYPE token's **force-quirks flag** to on. Switch to the **data state**. Emit that DOCTYPE token.

This is an **eof-in-doctype** parse error. Set the DOCTYPE token's **force-quirks flag** to on. Emit that DOCTYPE token. Emit an end-of-file token.

Append the current input character to the current DOCTYPE token's system identifier.
the data state. Emit that DOCTYPE token.

↩ EOF
This is an eof-in-doctype parse error. Set the DOCTYPE token's force-quirks flag to on. Emit that DOCTYPE token. Emit an end-of-file token.

↩ Anything else
Append the current input character to the current DOCTYPE token's system identifier.

13.2.5.67 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 data state. Emit the current DOCTYPE token.

↩ EOF
This is an eof-in-doctype parse error. Set the DOCTYPE token's force-quirks flag to on. Emit that DOCTYPE token. Emit an end-of-file token.

↩ Anything else
This is an unexpected-character-after-doctype-system-identifier parse error. Reconsume in the bogus DOCTYPE state. (This does not set the DOCTYPE token's force-quirks flag to on.)

13.2.5.68 Bogus DOCTYPE state

Consume the next input character:

↩ U+003E GREATER-THAN SIGN (>)
  Switch to the data state. Emit the DOCTYPE token.

↩ U+0000 NULL
  This is an unexpected-null-character parse error. Ignore the character.

↩ EOF
  Emit the DOCTYPE token. Emit an end-of-file token.

↩ Anything else
  Ignore the character.

13.2.5.69 CDATA section state

Consume the next input character:

↩ U+005D RIGHT SQUARE BRACKET (]
  Switch to the CDATA section bracket state.

↩ EOF
This is an eof-in-cdata parse error. Emit an end-of-file token.

↩ Anything else
  Emit the current input character as a character token.
13.2.5.70 CDATA section bracket state

Consume the next input character:

* U+005D RIGHT SQUARE BRACKET (】)
  
  Switch to the CDATA section end state.

* Anything else
  
  Emit a U+005D RIGHT SQUARE BRACKET character token. Reconsume in the CDATA section state.

13.2.5.71 CDATA section end state

Consume the next input character:

* U+005D RIGHT SQUARE BRACKET (】)
  
  Emit a U+005D RIGHT SQUARE BRACKET character token.

* U+003E GREATER-TYN SIGN character
  
  Switch to the data state.

* Anything else
  
  Emit two U+005D RIGHT SQUARE BRACKET character tokens. Reconsume in the CDATA section state.

13.2.5.72 Character reference state

Set the temporary buffer to the empty string. Append a U+0026 AMPERSAND (&) character to the temporary buffer. Consume the next input character:

* ASCII alphanumeric
  
  Reconsume in the named character reference state.

* U+0023 NUMBER SIGN (#)
  
  Append the current input character to the temporary buffer. Switch to the numeric character reference state.

* Anything else
  
  Flush code points consumed as a character reference. Reconsume in the return state.

13.2.5.73 Named character reference state

Consume the maximum number of characters possible, where the consumed characters are one of the identifiers in the first column of the named character references table. Append each character to the temporary buffer when it's consumed.

* If there is a match
  
  If the character reference was consumed as part of an attribute, and the last character matched is not a U+003B SEMICOLON character (;), and the next input character is either a U+003D EQUALS SIGN character (=) or an ASCII alphanumeric, then, for historical reasons, flush code points consumed as a character reference and switch to the return state.

  Otherwise:

  1. If the last character matched is not a U+003B SEMICOLON character (;), then this is a missing-semicolon-after-character-reference parse error.
2. Set the temporary buffer to the empty string. Append one or two characters corresponding to the character reference name (as given by the second column of the named character references table) to the temporary buffer.

3. Flush code points consumed as a character reference. Switch to the return state.

- Otherwise
  Flush code points consumed as a character reference. Switch to the ambiguous ampersand state.

Example

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 &notin; I tell you, the character reference would be parsed as "notin," resulting in I'm I tell you (and no parse error).

However, if the markup contains the string I'm &notit; I tell you in an attribute, no character reference is parsed and string remains intact (and there is no parse error).

13.2.5.74 Ambiguous ampersand state

Consume the next input character:

- ASCII alphanumeric
  If the character reference was consumed as part of an attribute, then append the current input character to the current attribute's value. Otherwise, emit the current input character as a character token.

- U+003B SEMICOLON (;)
  This is an unknown-named-character-reference parse error. Reconsume in the return state.

- Anything else
  Reconsume in the return state.

13.2.5.75 Numeric character reference state

Set the character reference code to zero (0).

Consume the next input character:

- U+0078 LATIN SMALL LETTER X
- U+0058 LATIN CAPITAL LETTER X
  Append the current input character to the temporary buffer. Switch to the hexadecimal character reference start state.

- Anything else
  Reconsume in the decimal character reference start state.

13.2.5.76 Hexadecimal character reference start state

Consume the next input character:

- ASCII hex digit
  Reconsume in the hexadecimal character reference state.

- Anything else
  This is an absence-of-digits-in-numeric-character-reference parse error. Flush code points consumed as a character reference. Reconsume in the return state.
13.2.5.77 Decimal character reference start state

Consume the next input character.

- **ASCII digit**
  - Recompute in the decimal character reference state.

- **Anything else**
  - This is an absence-of-digits-in-numeric-character-reference parse error. Flush code points consumed as a character reference. Recompute in the return state.

13.2.5.78 Hexadecimal character reference state

Consume the next input character:

- **ASCII digit**
  - Multiply the character reference code by 16. Add a numeric version of the current input character (subtract 0x0030 from the character's code point) to the character reference code.

- **ASCII upper hex digit**
  - Multiply the character reference code by 16. Add a numeric version of the current input character as a hexadecimal digit (subtract 0x0037 from the character's code point) to the character reference code.

- **ASCII lower hex digit**
  - Multiply the character reference code by 16. Add a numeric version of the current input character as a hexadecimal digit (subtract 0x0057 from the character's code point) to the character reference code.

- **U+003B SEMICOLON**
  - Switch to the numeric character reference end state.

- **Anything else**
  - This is a missing-semicolon-after-character-reference parse error. Recompute in the numeric character reference end state.

13.2.5.79 Decimal character reference state

Consume the next input character:

- **ASCII digit**
  - Multiply the character reference code by 10. Add a numeric version of the current input character (subtract 0x0030 from the character's code point) to the character reference code.

- **U+003B SEMICOLON**
  - Switch to the numeric character reference end state.

- **Anything else**
  - This is a missing-semicolon-after-character-reference parse error. Recompute in the numeric character reference end state.

13.2.5.80 Numeric character reference end state

Check the character reference code:

- If the number is 0x00, then this is a null-character-reference parse error. Set the character reference code to 0xFFFD.

- If the number is greater than 0x10FFFF, then this is a character-reference-outside-unicode-range parse error. Set the character reference code to 0xFFFD.

- If the number is a surrogate, then this is a surrogate-character-reference parse error. Set the character reference code.
• If the number is a noncharacter, then this is a noncharacter-character-reference\textsuperscript{1073} parse error\textsuperscript{1073}.
• If the number is 0x0D, or a control that’s not ASCII whitespace, then this is a control-character-reference\textsuperscript{1073} parse error\textsuperscript{1073}.
If the number is one of the numbers in the first column of the following table, then find the row with that number in the first column, and set the character reference code\textsuperscript{1115} to the number in the second column of that row.

<table>
<thead>
<tr>
<th>Number</th>
<th>Code point</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80</td>
<td>0x20AC</td>
</tr>
<tr>
<td>0x82</td>
<td>0x201A</td>
</tr>
<tr>
<td>0x83</td>
<td>0x0192</td>
</tr>
<tr>
<td>0x84</td>
<td>0x201E</td>
</tr>
<tr>
<td>0x85</td>
<td>0x2026</td>
</tr>
<tr>
<td>0x86</td>
<td>0x2020</td>
</tr>
<tr>
<td>0x87</td>
<td>0x2021</td>
</tr>
<tr>
<td>0x88</td>
<td>0x02C6</td>
</tr>
<tr>
<td>0x89</td>
<td>0x2030</td>
</tr>
<tr>
<td>0x8A</td>
<td>0x0160</td>
</tr>
<tr>
<td>0x8B</td>
<td>0x2039</td>
</tr>
<tr>
<td>0x8C</td>
<td>0x0152</td>
</tr>
<tr>
<td>0x8E</td>
<td>0x017D</td>
</tr>
<tr>
<td>0x91</td>
<td>0x2018</td>
</tr>
<tr>
<td>0x92</td>
<td>0x2019</td>
</tr>
<tr>
<td>0x93</td>
<td>0x201C</td>
</tr>
<tr>
<td>0x94</td>
<td>0x201D</td>
</tr>
<tr>
<td>0x95</td>
<td>0x2022</td>
</tr>
<tr>
<td>0x96</td>
<td>0x2013</td>
</tr>
<tr>
<td>0x97</td>
<td>0x2014</td>
</tr>
<tr>
<td>0x98</td>
<td>0x02DC</td>
</tr>
<tr>
<td>0x99</td>
<td>0x2022</td>
</tr>
<tr>
<td>0x9A</td>
<td>0x0161</td>
</tr>
<tr>
<td>0x9B</td>
<td>0x203A</td>
</tr>
<tr>
<td>0x9C</td>
<td>0x0153</td>
</tr>
<tr>
<td>0x9E</td>
<td>0x017E</td>
</tr>
<tr>
<td>0x9F</td>
<td>0x0178</td>
</tr>
</tbody>
</table>

Set the temporary buffer\textsuperscript{1089} to the empty string. Append a code point equal to the character reference code\textsuperscript{1115} to the temporary buffer\textsuperscript{1089}. Flush code points consumed as a character reference\textsuperscript{1089}. Switch to the return state\textsuperscript{1089}.

### 13.2.6 Tree construction

The input to the tree construction stage is a sequence of tokens from the tokenization\textsuperscript{1089} stage. The tree construction stage is associated with a DOM Document\textsuperscript{115} 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\textsuperscript{115} 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 a MathML annotation-xml element 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:

- A MathML mi element
- A MathML mo element
- A MathML mn element
- A MathML ms element
- A MathML mtext element

A node is an HTML integration point if it is one of the following elements:

- A MathML annotation-xml element whose start tag token had an attribute with the name "encoding" whose value was an ASCII case-insensitive match for the string "text/html"
- A MathML annotation-xml element 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"
- An SVG foreignObject element
- An SVG desc element
- An SVG title element

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 implementers are encouraged to avoid arbitrary limits, it is recognized that practical concerns will likely force user agents to impose nesting depth constraints.

13.2.6.1 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

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 steps.
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 steps. (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 steps.
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).

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 UA to create an element for a token in a particular given namespace and with a particular intended parent, the UA must run the following steps:

1. Let document be intended parent's node document.
2. Let local name be the tag name of the token.
3. Let is be the value of the "is" attribute in the given token, if such an attribute exists, or null otherwise.
4. Let definition be the result of looking up a custom element definition given document, given namespace, local name, and is.
5. If definition is non-null and the parser was not created as part of the HTML fragment parsing algorithm, then let will execute script be true. Otherwise, let it be false.
6. If will execute script is true, then:
   1. Increment document's throw-on-dynamic-markup-insertion counter.
   2. If the JavaScript execution context stack is empty, then perform a microtask checkpoint.
   3. Push a new element queue onto document's relevant agent's custom element reactions stack.
7. Let element be the result of creating an element given document, localName, given namespace, null, and is. If will execute script is true, set the synchronous custom elements flag; otherwise, leave it unset.
8. Append each attribute in the given token to element.

Note

This will cause custom element constructors\textsuperscript{704} to run, if will execute script is true. However, since we incremented the throw-on-dynamic-markup-insertion counter\textsuperscript{994}, this cannot cause new characters to be inserted into the tokenizer\textsuperscript{987}, or the document to be blown away\textsuperscript{996}.

9. If will execute script is true, then:
   1. Let queue be the result of popping from document's relevant agent\textsuperscript{900}'s custom element reactions stack\textsuperscript{713}. (This will be the same element queue\textsuperscript{713} as was pushed above.)
   2. Invoke custom element reactions\textsuperscript{714} in queue.
   3. Decrement document's throw-on-dynamic-markup-insertion counter\textsuperscript{994}.

10. If 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\textsuperscript{973}. Similarly, if element has an xmlns:xlink attribute in the XMLNS namespace whose value is not the XLink Namespace, that is a parse error\textsuperscript{973}.

11. If element is a resettable element\textsuperscript{486}, invoke its reset algorithm\textsuperscript{603}. (This initializes the element's value\textsuperscript{566} and checkedness\textsuperscript{566} based on the element's attributes.)

12. If element is a form-associated element\textsuperscript{486} and not a form-associated custom element\textsuperscript{9705}, the form element pointer\textsuperscript{1088} is not null, there is no template\textsuperscript{629} element on the stack of open elements\textsuperscript{987}, element is either not listed\textsuperscript{486} or doesn't have a form\textsuperscript{566} attribute, and the intended parent is in the same tree as the element pointed to by the form element pointer\textsuperscript{1088}, then associate\textsuperscript{566} element with the form\textsuperscript{1486} element pointed to by the form element pointer\textsuperscript{1088} and set element's parser inserted flag\textsuperscript{566}.

13. Return 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\textsuperscript{9118}.

2. Let element be the result of creating an element for the token\textsuperscript{9119} 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 element at the adjusted insertion location, then:
   1. If the parser was not created as part of the HTML fragment parsing algorithm\textsuperscript{9118}, then push a new element queue\textsuperscript{713} onto element's relevant agent\textsuperscript{900}'s custom element reactions stack\textsuperscript{713}.
   2. Insert element at the adjusted insertion location.
   3. If the parser was not created as part of the HTML fragment parsing algorithm\textsuperscript{9118}, then pop the element queue\textsuperscript{713} from element's relevant agent\textsuperscript{900}'s custom element reactions stack\textsuperscript{713}, and invoke custom element reactions\textsuperscript{714} in that queue.

Note

If the adjusted insertion location cannot accept more elements, e.g. because it's a Document\textsuperscript{9115} that already has an element child, then element is dropped on the floor.

4. Push element onto the stack of open elements\textsuperscript{9108} so that it is the new current node\textsuperscript{9108}.

5. Return 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.)

<table>
<thead>
<tr>
<th>Attribute name on token</th>
<th>Attribute name on element</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributename</td>
<td>attributeName</td>
</tr>
<tr>
<td>attributeType</td>
<td>attributeType</td>
</tr>
<tr>
<td>basefrequency</td>
<td>baseFrequency</td>
</tr>
<tr>
<td>basestyle</td>
<td>baseStyle</td>
</tr>
<tr>
<td>calcmode</td>
<td>calcMode</td>
</tr>
<tr>
<td>clippathunits</td>
<td>clipPathUnits</td>
</tr>
<tr>
<td>diffuseconstant</td>
<td>diffuseConstant</td>
</tr>
<tr>
<td>edgemode</td>
<td>edgeMode</td>
</tr>
<tr>
<td>filterunits</td>
<td>filterUnits</td>
</tr>
<tr>
<td>glyphref</td>
<td>glyphRef</td>
</tr>
<tr>
<td>gradienttransform</td>
<td>gradientTransform</td>
</tr>
<tr>
<td>gradientunits</td>
<td>gradientUnits</td>
</tr>
<tr>
<td>kernelmatrix</td>
<td>kernelMatrix</td>
</tr>
<tr>
<td>kernelunitlength</td>
<td>kernelUnitLength</td>
</tr>
<tr>
<td>keypoints</td>
<td>keyPoints</td>
</tr>
<tr>
<td>keyspilines</td>
<td>keySplines</td>
</tr>
<tr>
<td>keytimes</td>
<td>keyTimes</td>
</tr>
<tr>
<td>lengthadjust</td>
<td>lengthAdjust</td>
</tr>
<tr>
<td>limitingconeangle</td>
<td>limitingConeAngle</td>
</tr>
<tr>
<td>markerheight</td>
<td>markerHeight</td>
</tr>
<tr>
<td>markerunits</td>
<td>markerUnits</td>
</tr>
<tr>
<td>markerwidth</td>
<td>markerWidth</td>
</tr>
<tr>
<td>maskcontentunits</td>
<td>maskContentPane</td>
</tr>
<tr>
<td>maskunits</td>
<td>maskUnits</td>
</tr>
<tr>
<td>numoctaves</td>
<td>numOctaves</td>
</tr>
<tr>
<td>pathlength</td>
<td>pathLength</td>
</tr>
<tr>
<td>patterncontentunits</td>
<td>patternContentPane</td>
</tr>
<tr>
<td>patterntransform</td>
<td>patternTransform</td>
</tr>
<tr>
<td>patternunits</td>
<td>patternUnits</td>
</tr>
<tr>
<td>pointsatx</td>
<td>pointsAtX</td>
</tr>
<tr>
<td>pointsaty</td>
<td>pointsAtY</td>
</tr>
<tr>
<td>pointsatZ</td>
<td>pointsAtZ</td>
</tr>
<tr>
<td>preservealpha</td>
<td>preserveAlpha</td>
</tr>
<tr>
<td>preserveaspectratio</td>
<td>preserveAspectRatio</td>
</tr>
<tr>
<td>primitiveunits</td>
<td>primitiveUnits</td>
</tr>
<tr>
<td>refx</td>
<td>refX</td>
</tr>
<tr>
<td>refy</td>
<td>refY</td>
</tr>
<tr>
<td>repeatcount</td>
<td>repeatCount</td>
</tr>
<tr>
<td>repeatdur</td>
<td>repeatDur</td>
</tr>
<tr>
<td>requiredextensions</td>
<td>requiredExtensions</td>
</tr>
<tr>
<td>requiredfeatures</td>
<td>requiredFeatures</td>
</tr>
<tr>
<td>specularconstant</td>
<td>specularConstant</td>
</tr>
<tr>
<td>specularexponent</td>
<td>specularExponent</td>
</tr>
<tr>
<td>spreadmethod</td>
<td>spreadMethod</td>
</tr>
<tr>
<td>startoffset</td>
<td>startOffset</td>
</tr>
<tr>
<td>stddeviation</td>
<td>stdDeviation</td>
</tr>
</tbody>
</table>
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.)

<table>
<thead>
<tr>
<th>Attribute name on token</th>
<th>Attribute name on element</th>
</tr>
</thead>
<tbody>
<tr>
<td>stitchtiles</td>
<td>stitchTiles</td>
</tr>
<tr>
<td>surfacescale</td>
<td>surfaceScale</td>
</tr>
<tr>
<td>systemlanguage</td>
<td>systemLanguage</td>
</tr>
<tr>
<td>tablevalues</td>
<td>tableValues</td>
</tr>
<tr>
<td>targetx</td>
<td>targetX</td>
</tr>
<tr>
<td>targety</td>
<td>targetY</td>
</tr>
<tr>
<td>textlength</td>
<td>textLength</td>
</tr>
<tr>
<td>viewBox</td>
<td>viewBox</td>
</tr>
<tr>
<td>viewtarget</td>
<td>viewTarget</td>
</tr>
<tr>
<td>xchannelselector</td>
<td>xChannelSelector</td>
</tr>
<tr>
<td>ychannelselector</td>
<td>yChannelSelector</td>
</tr>
<tr>
<td>zoomandpan</td>
<td>zoomAndPan</td>
</tr>
</tbody>
</table>

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 return.
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.

Note

The DOM will not let Document nodes have Text node children, so they are dropped on the floor.

Example

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.

<table>
<thead>
<tr>
<th>Input</th>
<th>Number of Text nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A&lt;script&gt;</td>
<td>One Text node in the document, containing “AB”.</td>
</tr>
<tr>
<td>var script = document.getElementsByTagName('script')[0];</td>
<td></td>
</tr>
</tbody>
</table>
When the steps below require the user agent to insert a comment while processing a comment token, optionally with an explicitly insertion 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 UA parsing the document. This includes the parsing of any content inserted using `document.write()` and `document.writeln()` calls. However, mutation observers do fire, as required by DOM.

13.2.6.2 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 RAWTEXT state; otherwise the algorithm invoked was the generic RCDATA element parsing algorithm, switch the tokenizer to the RCDATA state.
3. Let the original insertion mode be the current insertion mode.
4. Then, switch the insertion mode to "text".
13.2.6.3 Closing elements that have implied end tags

When the steps below require the UA to **generate implied end tags**, then, while the current node\[^{1086}\] is a `dd` element, a `dt` element, an `li` element, an `optgroup` element, an `option` element, a `p` element, an `rtf` element, an `rtc` element, or an `rtt` element, the UA must pop the current node\[^{1086}\] off the stack of open elements\[^{1085}\].

If a step requires the UA to generate implied end tags but lists an element to exclude from the process, then the UA must perform the above steps as if that element was not in the above list.

When the steps below require the UA to **generate all implied end tags thoroughly**, then, while the current node\[^{1086}\] 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 `rtf` element, an `rtc` element, an `rtt` element, a `tbody` element, a `tfoot` element, an `thead` element, or a `tr` element, the UA must pop the current node\[^{1086}\] off the stack of open elements\[^{1085}\].

### 13.2.6.4 The rules for parsing tokens in HTML content

#### 13.2.6.4.1 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\[^{1123}\] as the last child of the `Document`\[^{115}\] object.

- **A DOCTYPE token**
  
  If the DOCTYPE token's name is not `html`, or the token's public identifier is not missing, or the token's system identifier is neither missing nor "about:legacy-compet"\[^{28}\], then there is a **parse error**\[^{1073}\].

  Append a `DocumentType` node to the `Document`\[^{115}\] 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`\[^{115}\] object so that it is returned as the value of the doctype attribute of the `Document`\[^{115}\] object.

  Then, if the document is not `an iframe srcdoc document`\[^{363}\], and the DOCTYPE token matches one of the conditions in the following list, then set the `Document`\[^{115}\] to **quirks mode**:

  - The force-quirs flag\[^{2088}\] is set to on.
  - The name is not `html`.
  - The public identifier is set to: "/W3C//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: "http://www.w3.org/MarkUp/DTD/xhtml1-transitional.dtd"
  - The public identifier starts with: "+//Silmarill//DTD html Pro v0r11 19970101/"
  - The public identifier starts with: "+//AS//DTD HTML 3.0 asWedit + extensions/"
  - The public identifier starts with: "//AdvSoft Ltd/DTD HTML 3.0 asWedit + extensions/"
  - The public identifier starts with: "//IE/IF/DTD HTML 2.0 Level 1/"
  - The public identifier starts with: "//IE/IF/DTD HTML 2.0 Level 2/"
  - The public identifier starts with: "//IE/IF/DTD HTML 2.0 Strict Level 1/"
  - The public identifier starts with: "//IE/IF/DTD HTML 2.0 Strict Level 2/"
  - The public identifier starts with: "//IE/IF/DTD HTML 2.0/"
  - The public identifier starts with: "//IE/IF/DTD HTML 2.1/"
  - The public identifier starts with: "//IE/IF/DTD HTML 3.0/"
  - The public identifier starts with: "//IE/IF/DTD HTML 3.2 Final/"
  - The public identifier starts with: "//IE/IF/DTD HTML 3.2/"
  - The public identifier starts with: "//IE/IF/DTD HTML 3/"
  - The public identifier starts with: "//IE/IF/DTD HTML Level 0/"
  - The public identifier starts with: "//IE/IF/DTD HTML Level 1/"
  - The public identifier starts with: "//IE/IF/DTD HTML Level 2/"
  - The public identifier starts with: "//IE/IF/DTD HTML Level 3/"
  - The public identifier starts with: "//IE/IF/DTD HTML Strict Level 0/"
  - The public identifier starts with: "//IE/IF/DTD HTML Strict Level 1/"
  - The public identifier starts with: "//IE/IF/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 HTML 1.0 Frameset/"
• The public identifier starts with: "-//W3C/DTD HTML 1.0 Transitional/"
• The public identifier starts with: "-//W3C/DTD XHTML 1.0 Transitional/"
• The public identifier starts with: "-//W3C/DTD XHTML 1.0 Frameset/"
• The public identifier starts with: "-//W3C/DTD XHTML Strict/"
• The public identifier starts with: "-//W3C/DTD XHTML+ Frameset/"
• The public identifier starts with: "-//W3C/DTD XHTML+ Transitional/"
• The public identifier starts with: "-//W3C/DTD HTML 2.0 Frameset/"
• The public identifier starts with: "-//W3C/DTD HTML 2.0 Transitional/"
• The public identifier starts with: "-//W3C/DTD HTML 2.0 Strict/"
• The public identifier starts with: "-//W3C/DTD 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 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/"

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.

13.2.6.4.2 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.

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

Switch the insertion mode to "before head", then reprocess the token.

The document 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.

13.2.6.4.3 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.
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. Set the element's parser document to the Document, and unset the element's "non-blocking" flag.

4. If the parser was created as part of the HTML fragment parsing algorithm, then mark the script element as

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.
5. If the parser was invoked via the `document.write()` or `document.writeln()` methods, then optionally mark the `script` element as "already started". (For example, the user agent might use this clause to prevent execution of cross-origin scripts inserted via `document.write()` under slow network conditions, or when the page has already taken a long time to load.)

6. Insert the newly created element at the adjusted insertion location.

7. Push the element onto the stack of open elements so that it is the new current node.

8. Switch the tokenizer to the `script data state`.

9. Let the original insertion mode be the current insertion mode.

10. 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.
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.

↪ A end tag whose tag name is "noscript"
 Pop 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
 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 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 start tag whose tag name is "head", "noscript"

↪ Any other end tag
 Parse error. Ignore the token.

↪ Anything else
 Parse error. Ignore the token.

Pop the stack of open elements; the new current node will be a head element.

Switch the insertion mode to "in head".

Reprocess the token.

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".

مى A start tag whose tag name is "frameset"

*Insert an HTML element* for the token.

Switch the *insertion mode* to "in frameset".

مى 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.

مى An end tag whose tag name is "template"

*Process the token using the rules for the "in head" insertion mode*.  

مى An end tag whose tag name is one of: "body", "html", "br"

Act as described in the "anything else" entry below.

مى A start tag whose tag name is "head"

مى Any other end tag

*Parse error*. Ignore the token.

مى 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.

### 13.2.6.4.7 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:

مى A character token that is U+0000 NULL

*Parse error*. Ignore the token.

مى 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*.  

*Set the frameset-ok flag to "not ok".*

مى 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 rt element, an tbody 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 rt element, a tbody element, a thead element, a tr element, the body element, or the html element, then this is a parse error.
A *tr* element, the *body* element, or the *html* element, then this is a *parse error*.

Switch the *insertion mode* to "*after body*".

.GetAxis

**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 *optgroup* element, an *option* element, a *p* element, an *rb* element, an *rp* element, an *rt* element, an *tbody* element, a *td* element, a *tfoot* element, a *th* element, an *thead* 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", "dialog", "dir", "div", "dl", "fieldset", "figcaption", "figure", "footer", "header", "hgroup", "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 *li* element, then run these substeps:
   1. Generate implied end tags except for *li* elements.
   2. If the current node is not an *li* 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 **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 **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\(^{1086}\), if any.
3. Insert an HTML element\(^{1121}\) for the token.
4. Set the frameset-ok flag\(^{1089}\) to "not ok".

An end tag whose tag name is one of: "address", "article", "aside", "blockquote", "button", "center", "details", "dialog", "dir", "div", "dl", "fieldset", "figcaption", "figure", "footer", "header", "hgroup", "listing", "main", "menu", "nav", "ol", "pre", "section", "summary", "ul"

If the stack of open elements\(^{1085}\) does not have an element in scope\(^{1086}\) that is an HTML element\(^{44}\) with the same tag name as that of the token, then this is a parse error\(^{1073}\); ignore the token.

Otherwise, run these steps:
1. Generate implied end tags\(^{1124}\).
2. If the current node\(^{1086}\) is not an HTML element\(^{44}\) with the same tag name as that of the token, then this is a parse error\(^{1073}\).
3. Pop elements from the stack of open elements\(^{1085}\) until an HTML element\(^{44}\) 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\(^{629}\) element on the stack of open elements\(^{1085}\), then run these substeps:
1. Let node be the element that the form element pointer\(^{1088}\) is set to, or null if it is not set to an element.
2. Set the form element pointer\(^{1088}\) to null.
3. If node is null or if the stack of open elements\(^{1085}\) does not have node in scope\(^{1086}\), then this is a parse error\(^{1073}\); return and ignore the token.
4. Generate implied end tags\(^{1124}\).
5. If the current node\(^{1086}\) is not node, then this is a parse error\(^{1073}\).
6. Remove node from the stack of open elements\(^{1085}\).

If there is a template\(^{629}\) element on the stack of open elements\(^{1085}\), then run these substeps instead:
1. If the stack of open elements\(^{1085}\) does not have a form element in scope\(^{1086}\), then this is a parse error\(^{1073}\); return and ignore the token.
2. Generate implied end tags\(^{1124}\).
3. If the current node\(^{1086}\) is not a form\(^{486}\) element, then this is a parse error\(^{1073}\).
4. Pop elements from the stack of open elements\(^{1085}\) until a form\(^{486}\) element has been popped from the stack.

An end tag whose tag name is "p"

If the stack of open elements\(^{1085}\) does not have a p element in button scope\(^{1087}\), then this is a parse error\(^{1073}\); insert an HTML element\(^{1121}\) for a "p" start tag token with no attributes.

Close a p element\(^{1139}\).

An end tag whose tag name is "li"

If the stack of open elements\(^{1085}\) does not have an li element in list item scope\(^{1087}\), then this is a parse error\(^{1073}\); ignore the token.

Otherwise, run these steps:
1. Generate implied end tags\(^{1124}\), except for li\(^{1025}\) elements.
2. If the current node\(^{1086}\) is not an li\(^{1025}\) element, then this is a parse error\(^{1073}\).
3. Pop elements from the stack of open elements\(^{1085}\) until an li\(^{1025}\) 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 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 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 with the same tag name as 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 token, then remove that element from the list of active formatting elements if the adoption agency algorithm didn't already remove it (it might not have if the element is not in table scope).

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 anobr element in scope, then this is a parse error; run the adoption agency algorithm for the token, 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.

Example

In the non-conforming stream `<a href="a">a</a><table><a href="b">b</a></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.
Run the adoption agency algorithm for the token.

Reconstruct the active formatting elements, if any.

Insert a marker at the end of the list of active formatting elements.

Set the frameset-ok flag to "not ok".

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".

Insert an HTML element for the token. Immediately pop the current node off the stack of open elements.

Insert an HTML element for the token. Immediately pop the current node off the stack of open elements.

Set the frameset-ok flag to "not ok".

Insert an HTML element for the token. Immediately pop the current node off the stack of open elements.

Switch the insertion mode to "in table".

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".

Insert an HTML element for the token. Immediately pop the current node off the stack of open elements.
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 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 now a rtc element or a ruby 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 return.
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 token token for which the algorithm is being run, consists of the following steps:

1. Let subject be token's tag name.
2. 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 return.
3. Let outer loop counter be zero.
4. Outer loop: If outer loop counter is greater than or equal to eight, then return.
5. Increment outer loop counter by one.
6. 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 return and instead act as described in the "any other end tag" entry above.
7. If formatting element is not in the stack of open elements, then this is a parse error; remove the element from the list, and return.
8. If formatting element is in the stack of open elements, but the element is not in scope, then this is a parse error; return.
9. If formatting element is not the current node, this is a parse error. (But do not return.)
10. 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.
11. If there is no furthest block, then the UA must first pop all the nodes from the bottom of the stack of open elements, up to and including formatting element, then remove formatting element from the list of active formatting elements, and finally return.
12. Let common ancestor be the element immediately above formatting element in the stack of open elements.
13. 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.
14. 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.
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.

15. 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.

16. Create an element for the token for which formatting element was created, in the HTML namespace, with furthest block as the intended parent.

17. Take all of the child nodes of furthest block and append them to the element created in the last step.

18. Append that new element to furthest block.

19. 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.

20. 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.


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.

13.2.6.4.8 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, 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, return.

  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.
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 UA to clear the stack back to a table context, it means that the UA 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.
13.2.6.4.10 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 ASCII whitespace, 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.

13.2.6.4.11 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.

13.2.6.4.12 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.

↪ An end 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.

13.2.6.4.13 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\(^{1073}\).

Clear the stack back to a table body context\(^{1146}\). (See below.)

Insert an HTML element\(^{1121}\) for a "tr" start tag token with no attributes, then switch the insertion mode\(^{1084}\) to "in row\(^{1146}\)."

Reprocess the current token.

An end tag whose tag name is one of: "tbody", "tfoot", "thead"

If the stack of open elements\(^{1085}\) does not have an element in table scope\(^{1087}\) that is an HTML element\(^{44}\) with the same tag name as the token, this is a parse error\(^{1073}\); ignore the token.

Otherwise:

Clear the stack back to a table body context\(^{1146}\). (See below.)

Pop the current node\(^{1086}\) from the stack of open elements\(^{1085}\). Switch the insertion mode\(^{1084}\) to "in table\(^{1142}\)."

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\(^{1085}\) does not have a tbody, thead, or tfoot element in table scope\(^{1087}\), this is a parse error\(^{1073}\); ignore the token.

Otherwise:

Clear the stack back to a table body context\(^{1146}\). (See below.)

Pop the current node\(^{1086}\) from the stack of open elements\(^{1085}\). Switch the insertion mode\(^{1084}\) to "in table\(^{1142}\)."

Reprocess the token.

An end tag whose tag name is one of: "body", "caption", "col", "colgroup", "html", "td", "th", "tr"

Parse error\(^{1073}\). Ignore the token.

Anything else

Process the token using the rules for the "in table\(^{1142}\)" insertion mode\(^{1084}\).

When the steps above require the UA to clear the stack back to a table body context, it means that the UA must, while the current node\(^{1086}\) is not a tbody\(^{461}\), tfoot\(^{463}\), thead\(^{462}\), template\(^{462}\), or html\(^{153}\) element, pop elements from the stack of open elements\(^{1085}\).

The current node\(^{1086}\) being an html\(^{153}\) element after this process is a fragment case\(^{1169}\).

13.2.6.4.14 The "in row" insertion mode\(^{11}\)

When the user agent is to apply the rules for the "in row\(^{1146}\)" insertion mode\(^{1084}\), 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\(^{1147}\). (See below.)

Insert an HTML element\(^{1121}\) for the token, then switch the insertion mode\(^{1084}\) to "in cell\(^{1147}\)."

Insert a marker\(^{1087}\) at the end of the list of active formatting elements\(^{1087}\).

An end tag whose tag name is "tr"

If the stack of open elements\(^{1085}\) does not have a tr element in table scope\(^{1087}\), this is a parse error\(^{1073}\); 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 UA to clear the stack back to a table row context, it means that the UA must, while the current node is not a tr, template, or html element, pop elements from the stack of open elements.

The current node being an html element after this process is a fragment case.

13.2.6.4.15 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\textsuperscript{11088}.

Switch the insertion mode\textsuperscript{11084} to “in row\textsuperscript{11146}.”

\textbf{↩} A start tag whose tag name is one of: “caption”, “col”, “colgroup”, “tbody”, “td”, “tfoot”, “th”, “thead”, “tr”

If the stack of open elements\textsuperscript{11085} does not have a td or th element in table scope\textsuperscript{11087}, then this is a parse error\textsuperscript{11073}; ignore the token. (fragment case\textsuperscript{11169})

Otherwise, close the cell\textsuperscript{11148} (see below) and reprocess the token.

\textbf{↩} An end tag whose tag name is one of: “body”, “caption”, “col”, “colgroup”, “html”

Parse error\textsuperscript{11073}. Ignore the token.

\textbf{↩} An end tag whose tag name is one of: “table”, “tbody”, “tfoot”, “thead”, “tr”

If the stack of open elements\textsuperscript{11085} does not have an element in table scope\textsuperscript{11087} that is an HTML element\textsuperscript{1144} with the same tag name as that of the token, then this is a parse error\textsuperscript{11073}; ignore the token.

Otherwise, close the cell\textsuperscript{11148} (see below) and reprocess the token.

\textbf{↩} Anything else

Process the token using the rules for the “in body\textsuperscript{11130}” insertion mode\textsuperscript{11084}.

Where the steps above say to close the cell, they mean to run the following algorithm:

1. Generate implied end tags\textsuperscript{11124}.
2. If the current node\textsuperscript{11086} is not now a td\textsuperscript{1146} element or a th\textsuperscript{1147} element, then this is a parse error\textsuperscript{11073}.
3. Pop elements from the stack of open elements\textsuperscript{11085} stack until a td\textsuperscript{1146} element or a th\textsuperscript{1147} element has been popped from the stack.
4. Clear the list of active formatting elements up to the last marker\textsuperscript{11088}.
5. Switch the insertion mode\textsuperscript{11084} to “in row\textsuperscript{11146}.”

\textbf{Note}

The stack of open elements\textsuperscript{11085} cannot have both a td\textsuperscript{1146} element and a th\textsuperscript{1147} element in table scope\textsuperscript{11087} at the same time, nor can it have neither when the close the cell\textsuperscript{11148} algorithm is invoked.

\textbf{13.2.6.4.16 The “in select” insertion mode} \textsuperscript{1148}

When the user agent is to apply the rules for the “in select\textsuperscript{11144}” insertion mode\textsuperscript{11084}, the user agent must handle the token as follows:

\textbf{↩} A character token that is U+0000 NULL

Parse error\textsuperscript{11073}. Ignore the token.

\textbf{↩} Any other character token

Insert the token’s character\textsuperscript{11122}.

\textbf{↩} A comment token

Insert a comment\textsuperscript{11123}.

\textbf{↩} A DOCTYPE token

Parse error\textsuperscript{11073}. Ignore the token.

\textbf{↩} A start tag whose tag name is “html”

Process the token using the rules for the “in body\textsuperscript{11130}” insertion mode\textsuperscript{11084}.

\textbf{↩} A start tag whose tag name is “option”

If the current node\textsuperscript{11086} is an option\textsuperscript{1154} element, pop that node from the stack of open elements\textsuperscript{11085}.
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"

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"

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.

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, 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 one of: "script", "template"

Process the token using the rules for the "in head" insertion mode.

An end tag whose tag name is "template"

Process the token using the rules for the "in body" 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.

13.2.6.4.17 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.

13.2.6.4.18 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.

13.2.6.4.19 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+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 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.

13.2.6.4.20 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 created as part of the HTML fragment parsing algorithm, 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.
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.

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
   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-of-file token
   Stop parsing.

↪ Anything else
   Parse error. Switch the insertion mode to "in body" and reprocess the token.

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.

13.2.6.5 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 "font", if the token has any attributes named "color", "face", or "size"
Parse error.
If the parser was created as part of 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
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 an SVG script element

  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 SVG `<script>` element according to the SVG rules, if the user agent supports SVG. [SVG]

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, then return. (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 return.
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.

13.2.7 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 step 1.
4. Queue a global task on the DOM manipulation task source given the Document's relevant global object to run the following substeps:
   1. Fire an event named DOMContentLoaded at the Document object, with its bubbles attribute initialized to...
true.

2. Enable the client message queue of the ServiceWorkerContainer object whose associated service worker client is the Document object's relevant settings object.

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 global task on the DOM manipulation task source given the Document's relevant global object to run the following substeps:

   1. Set the current document readiness to "complete".
   2. Load event: If the Document object's browsing context is non-null, then fire an event named load at the Document object's relevant global object, with legacy target override flag set.

8. If the Document object's browsing context is non-null, then queue a global task on the DOM manipulation task source given the Document's relevant global object to run these steps:

   1. If the Document's page showing flag is true, then return (i.e. don't fire the event below).
   2. Set the Document's page showing flag to true.
   3. Fire an event named pageshow at the Document object's relevant global object, using PageTransitionEvent, with the persisted attribute initialized to false, and legacy target override flag set.

9. If the Document's print when loaded flag is set, then run the printing steps.

10. The Document is now ready for post-load tasks.


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".

13.2.8 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 tool chain 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 code point when expressed in hexadecimal, using digits 0-9 and capital letters A-F as the symbols, in increasing numeric order.
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.

This section is non-normative.

This section examines some erroneous markup and discusses how the HTML parser handles these cases.

### 13.2.9 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.

#### 13.2.9.1 Misnested tags: <b><i></b></i> §115

This section is non-normative.

The most-often discussed example of erroneous markup is as follows:

```html
<p>1<b>2</i>3</b>4</p>
```

The parsing of this markup is straightforward up to the "3". At this point, the DOM looks like this:

```html
<head>
    <html>
        <head>
            <body>
                <p>1<b>2</i>3</b>4</p>
            </body>
        </head>
    </html>
</head>
```
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:

```
  html
  |   head
  |     body
  |     p
      #text: 1
      b
      #text: 2
      i
      #text: 3
      #text: 4
      #text: 5
```

13.2.9.2 Misnested tags: `<b><p></b></p>`

This section is non-normative.

A case similar to the previous one is the following:

```
  <b>1</b><p>2</p><b>3</b>
```

Up to the "2" the parsing here is straightforward:

```
  html
  |   head
  |     body
  |     b
      #text: 1
      p
      #text: 2
```

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\textsuperscript{1087}, but since that list has only one element in it, the bookmark won’t have much effect.

As the algorithm progresses, \textit{node} ends up set to the formatting element (\texttt{b}\texttt{p274}), and \textit{last node} ends up set to the furthest block (\texttt{p}\texttt{p213}).

The \textit{last node} gets appended (moved) to the \textit{common ancestor}, so that the DOM looks like:

```
html
  head
  body
    b\texttt{p274}
    p\texttt{p213}
```

A new \texttt{b}\texttt{p274} element is created, and the children of the \texttt{p}\texttt{p213} element are moved to it:

```
html
  head
  body
    b\texttt{p274}
    p\texttt{p213}
      b\texttt{p274}
```

Finally, the new \texttt{b}\texttt{p274} element is appended to the \texttt{p}\texttt{p213} element, so that the DOM looks like:

```
html
  head
  body
    b\texttt{p274}
    p\texttt{p213}
      b\texttt{p274}
```

The \texttt{b}\texttt{p274} element is removed from the \textit{list of active formatting elements}\textsuperscript{1087} and the \textit{stack of open elements}\textsuperscript{1085}, so that when the “3” is parsed, it is appended to the \texttt{p}\texttt{p213} element:

```
html
  head
  body
    b\texttt{p274}
    p\texttt{p213}
      b\texttt{p274}
      #text: 2
```

13.2.9.3 Unexpected markup in tables \textsuperscript{911}

\textit{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><td>bbb</td></table>ccc
```

The highlighted \texttt{b}\texttt{p274} element start tag is not allowed directly inside a table like that, and the parser handles this case by placing the element \textit{before} the table. (This is called \textit{foster parenting}\textsuperscript{1012}.) This can be seen by examining the DOM tree as it stands just after the \texttt{table}\textsuperscript{565} element’s start tag has been seen:
...and then immediately after the <b> element start tag has been seen:

```html
<html><body><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 `b` element in it; and the insertion mode is "in table".

The `tr` start tag causes the `b` 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:

```html
<html><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 `b` 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).

The marker means that when the "aaa" character tokens are seen, no `b` element is created to hold the resulting Text node:

```html
<html><body><b><table><tbody><tr><td>
```

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 `b` 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
 insertion mode, which defer to the "in body" insertion mode but with foster parenting.

When the active formatting elements are reconstructed, a `b` element is created and foster parented, and then the "bbb" Text node is appended to it:

```
<html>

<head>

<body>

  b

#text: bbb
table
tbody
  tr
    td
      #text: aaa

</tbody>
</table>
</body>
</head>
</html>
```

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 `b` element in it; and the insertion mode is still "in table body".

Had the character tokens been only ASCII whitespace instead of "bbb", then that ASCII whitespace 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 `b` element being created, this time after the table:

```
<html>

<head>

<body>

  b

#text: bbb
table
tbody
t
  tr
    td
      #text: aaa

#text: ccc
</tbody>
</table>
</body>
</head>
</html>
```

### 13.2.9.4 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:

```html
<div id=a>
  <script>
    var div = document.getElementById('a');
    parent.document.body.appendChild(div);
  </script>
  alert(document.URL);
</div>
```
Up to the first "script" end tag, before the script is parsed, the result is relatively straightforward:

```
<html>
<head></head>
<body>
  <div id="a">
    <script>
      var div = document.getElementById('a');
      parent.document.body.appendChild(div);
    </script>
  </div>
</body>
</html>
```

After the script is parsed, though, the `div` element and its child `script` element are gone:

```
<html>
<head></head>
<body>
</body>
</html>
```

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 `Document`s 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`:

```
<html>
<head></head>
<body>
  <script>
    alert(document.URL);
  </script>
</body>
</html>
```

This script does execute, resulting in an alert that says "https://example.com/inner".

### 13.2.9.5 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 `Document`s is considered a bad practice.

### 13.2.9.6 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.

```html
<!DOCTYPE html>
<p><b class=x><b class=x><b class=x><b class=x><b class=x><b>x</b></b></b></b>
</p>
```

1163
The resulting DOM tree is as follows:

```
DOCTYPE: html

html
  head
  body

  p
    b
      class="x"
      class="x"
      class="x"
    b
      class="x"
      class="x"
      class="x"
    b
      class="x"
      class="x"
      class="x"
  #text: X

  p
    b
      class="x"
      class="x"
      class="x"
    b
      class="x"
      class="x"
      class="x"
  #text: X

  p
    b
      class="x"
      class="x"
      class="x"
    b
      class="x"
      class="x"
      class="x"
  #text: X

  p
    #text: X
```

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.

### 13.3 Serializing HTML fragments

For the purposes of the following algorithm, an element serializes as void if its element type is one of the void elements, or is basefont, basound, frame, or keygen.

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 returns a string.

**Note**

This algorithm serializes the children of the node being serialized, not the node itself.

1. If the node serializes as void, then return the empty string.
2. Let s be a string, and initialize it to the empty string.
3. If the node is a template element, then let the node instead be the template element’s template contents (a
4. 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\textsuperscript{44} created by the HTML parser\textsuperscript{1071} or createElement(), *tagname* will be lowercase.

      If *current node*’s *is value* is not null, and the element does not have an *is*\textsuperscript{9704} attribute in its attribute list, then append the string " *is="", followed by *current node*’s *is value escaped as described below\textsuperscript{1168} in attribute mode*, followed by a U+0022 QUOTATION MARK character (").

      For each attribute that the element has, append a U+0020 SPACE character, the attribute’s serialized name as described below\textsuperscript{1165}, a U+003D EQUALS SIGN character (=), a U+0022 QUOTATION MARK character ("), the attribute’s value, escaped as described below\textsuperscript{1168} 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\textsuperscript{44} set by the HTML parser\textsuperscript{1071} or 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 implementation-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-THERE SIGN character (>).

      If *current node* serializes as void\textsuperscript{1164}, then continue on to the next child node at this point.

      Append the value of running the HTML fragment serialization algorithm\textsuperscript{1164} 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-THERE SIGN character (>).
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).

5. Return s.

⚠ Warning!

It is possible that the output of this algorithm, if parsed with an HTML parser, will not return the original tree structure. Tree structures that do not roundtrip a serialize and reparse step can also be produced by the HTML parser itself, although such cases are typically non-conforming.

Example
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 control. 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.

Example
For example, consider the following markup:

```html
<form id="outer"><div></form><form id="inner"><input>
```

This will be parsed into:

```
html
   head
      body
         form id="outer"
            div
         form id="inner"
            input
```
The `<input>` element will be associated with the inner `<form>` element. Now, if this tree structure is serialized and reparsed, the `<form id="inner">` start tag will be ignored, and so the `<input>` element will be associated with the outer `<form>` element instead.

```html
<html>
  <head></head>
  <body><form id="outer"><div><form id="inner"><input></form></div></form></body>
</html>
```

**Example**

As another example, consider the following markup:

```html
<a><a><table></a></table></a>
```

This will be parsed into:

```html
<html>
  <head></head>
  <body>
    <a><a></a></a><table></table>
  </body>
</html>
```

That is, the `<a>` elements are nested, because the second `<a>` element is foster parented. After a serialize-reparse roundtrip, the `<a>` elements and the `<table>` element would all be siblings, because the second `<a>` start tag implicitly closes the first `<a>` element.

```html
<html>
  <head></head>
  <body>
    <a><a></a></a><table/></table></a></body></html>
```

For historical reasons, this algorithm does not round-trip an initial U+000A LINE FEED (LF) character in `<pre>`, `<textarea>`, or `<listing>` elements, even though (in the first two cases) the markup being round-tripped can be conforming. The HTML parser will drop such a character during parsing, but this algorithm does not serialize an extra U+000A LINE FEED (LF) character.

**Example**

For example, consider the following markup:

```html
<pre>
Hello.</pre>
```

When this document is first parsed, the `<pre>` element's child text content starts with a single newline character. After a serialize-reparse roundtrip, the `<pre>` element's child text content is simply "Hello.".

Because of the special role of the `<is>` attribute in signaling the creation of customized built-in elements, in that it provides a mechanism for parsed HTML to set the element's `is` value, we special-case its handling during serialization. This ensures that an element's `is` value is preserved through serialize-parse roundtrips.
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 "&amp;".
2. Replace any occurrences of the U+00A0 NO-BREAK SPACE character by the string "&nbsp;".
3. If the algorithm was invoked in the attribute mode, replace any occurrences of the "=" character by the string "&quot;.
4. If the algorithm was not invoked in the attribute mode, replace any occurrences of the "<" character by the string "&lt;", and any occurrences of the ">" character by the string ">".

13.4 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.
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, switching on the context element:
   - title
   - textarea
     → Switch the tokenizer to the RCDATA state.
   - style
   - xmp
   - iframe
   - noembed
   - noframes
     → Switch the tokenizer to the RAWTEXT state.
   - script
     → Switch the tokenizer to the script data state.
   - noscript
     → If the scripting flag is enabled, switch the tokenizer to the RAWTEXT state. Otherwise, leave the tokenizer in the data state.
   - plaintext
     → Switch the tokenizer to the PLAINTEXT state.
   - Any other element
     → Leave the tokenizer in the data state.

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 element and whose attributes are the attributes of context element.

10. Reset the parser's insertion mode appropriately.

**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.

**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, including the element itself, if it is a `form element`. If there is no such `form element`, 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`.

### 13.5 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.

**Note**

It is intentional, for legacy compatibility, that many code points have multiple character reference names. For example, some appear both with and without the trailing semicolon, or with different capitalizations.
This data is also available as a JSON file.

The glyphs displayed above are non-normative. Refer to Unicode for formal definitions of the characters listed above.

Note

The character reference names originate from XML Entity Definitions for Characters, though only the above is considered normative. [XMLENTITY]12778
14 The XML 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".

14.1 Writing documents in the XML syntax

Note
The XML syntax for HTML was formerly referred to as "XHTML", but this specification does not use that term (among other reasons, because no such term is used for the HTML syntaxes of MathML and SVG).

The syntax for XML is defined in XML and Namespaces in XML. [XML] [XMLNS]

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 XML, 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 XML documents is unsafe if they are defined in an external file (except for &lt;, &gt;, &amp;, &quot; and &apos;).

14.2 Parsing XML 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 XML 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 XML, Namespaces in XML, and DOM. When creating DOM nodes representing elements, the create an element for a token algorithm or some equivalent that operates on appropriate XML datastructures must be used, to ensure the proper element interfaces are created and that custom elements are set up correctly.

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 DOM's requirements regarding mutation observers mean that mutation observers are fired (unlike mutation events). [XML] [XMLNS] [XMLNS] [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.
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 named character references section.)

- //W3C/DTD XHTML 1.0 Transitional//EN
- //W3C/DTD XHTML 1.1//EN
- //W3C/DTD XHTML 1.0 Strict//EN
- //W3C/DTD XHTML 1.0 Frameset//EN
- //W3C/DTD XHTML Basic 1.0//EN
- //W3C/DTD XHTML 1.1 plus MathML 2.0//EN
- //W3C/DTD XHTML 1.1 plus MathML 2.0 plus SVG 1.1//EN
- //W3C/DTD MathML 2.0//EN
- //WAPFORUM/DTD XHTML 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.

XML parsers can be invoked with XML scripting support enabled or XML scripting support 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 have its parser document set and its "non-blocking" flag must be unset. If the parser was created as part of 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.

XML parsers would append a node to a template element, it must instead append it to the template element's template contents (a DocumentFragment node).

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 XML syntax, then it is an XML document.

### 14.3 Serializing XML 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 Document's, 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] [XMLNS]

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" DOMException 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". [XMLNS]
- 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 (:)....
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.
6. If the document element of the resulting Document has any sibling nodes, then throw a "SyntaxError" DOMException.
7. Return the child nodes of the document element of the resulting Document, in tree order.

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" DOMException.
15 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, "must" has 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 "must."

15.1 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 implementers 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.

An element is said to intersect the viewport when it is being rendered and its associated CSS layout box intersects the viewport.

Note

This specification does not define the precise timing for when the intersection is tested, but it is suggested that the timing match that of the Intersection Observer API.

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.

Note

This is especially important for issues relating to the 'display', 'unicode-bidi', and 'direction' properties.

15.2 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.
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 nonzero 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 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-inline-start’ and ‘margin-inline-end’ 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 margins on the **line-left** and **line-right** sides be set accordingly. [CSSLOGICAL] [CSSWM]

### 15.3 Non-replaced elements

#### 15.3.1 Hidden elements

```css
@namespace url(http://www.w3.org/1999/xhtml);

[hidden], area, base, basefont, datalist, head, link, meta, noembed, noframes, param, rp, script, source, style, template, track, title {
  display: none;
}

embed[hidden] { display: inline; height: 0; width: 0; }

input[type=hidden] { display: none !important; }

@media (scripting) {
  noscript { display: none !important; }
}
```

#### 15.3.2 The page

```css
@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 a default value of 8px is expected to be used for that property instead.

<table>
<thead>
<tr>
<th>Property</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>margin-top</code></td>
<td>The <strong>body</strong> element’s <strong>marginheight</strong> attribute&lt;br&gt;The <strong>body</strong> element’s <strong>topmargin</strong> attribute&lt;br&gt;The <strong>body</strong> element’s <strong>container frame element</strong>’s <strong>marginheight</strong> attribute</td>
</tr>
</tbody>
</table>

For each property in the table below, given a **body** element, the first attribute that exists **maps to the dimension 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 a default value of 8px is expected to be used for that property instead.

<table>
<thead>
<tr>
<th>Property</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>margin-top</code></td>
<td>The <strong>body</strong> element’s <strong>marginheight</strong> attribute&lt;br&gt;The <strong>body</strong> element’s <strong>topmargin</strong> attribute&lt;br&gt;The <strong>body</strong> element’s <strong>container frame element</strong>’s <strong>marginheight</strong> attribute</td>
</tr>
</tbody>
</table>

For each property in the table below, given a **body** element, the first attribute that exists **maps to the dimension property (ignoring zero)** 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 a default value of 8px is expected to be used for that property instead.

<table>
<thead>
<tr>
<th>Property</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>margin-top</code></td>
<td>The <strong>body</strong> element’s <strong>marginheight</strong> attribute&lt;br&gt;The <strong>body</strong> element’s <strong>topmargin</strong> attribute&lt;br&gt;The <strong>body</strong> element’s <strong>container frame element</strong>’s <strong>marginheight</strong> attribute</td>
</tr>
</tbody>
</table>

For each property in the table below, given a **body** element, the first attribute that exists **maps to the dimension property (ignoring zero)** 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 a default value of 8px is expected to be used for that property instead.

<table>
<thead>
<tr>
<th>Property</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>margin-top</code></td>
<td>The <strong>body</strong> element’s <strong>marginheight</strong> attribute&lt;br&gt;The <strong>body</strong> element’s <strong>topmargin</strong> attribute&lt;br&gt;The <strong>body</strong> element’s <strong>container frame element</strong>’s <strong>marginheight</strong> attribute</td>
</tr>
<tr>
<td>Property</td>
<td>Source</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>'margin-right'</td>
<td>The body element's margin-right attribute</td>
</tr>
<tr>
<td></td>
<td>The body element's right attribute</td>
</tr>
<tr>
<td></td>
<td>The body element's container frame element's margin-right attribute</td>
</tr>
<tr>
<td>'margin-bottom'</td>
<td>The body element's margin-bottom attribute</td>
</tr>
<tr>
<td></td>
<td>The body element's bottom attribute</td>
</tr>
<tr>
<td></td>
<td>The body element's container frame element's margin-bottom attribute</td>
</tr>
<tr>
<td>'margin-left'</td>
<td>The body element's margin-left attribute</td>
</tr>
<tr>
<td></td>
<td>The body element's left attribute</td>
</tr>
<tr>
<td></td>
<td>The body element's container frame element's margin-left attribute</td>
</tr>
</tbody>
</table>

If the body element's node document's browsing context is a child browsing context, and the container of that 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 or iframe element. 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.

⚠️ Warning!

If a Document's browsing context is a child browsing context, then it is expected to be positioned and sized to fit inside the content box of the container of that browsing context. If the container is not being rendered, the browsing context is expected to have a viewport with zero width and zero height.

If a Document's browsing context is a child browsing context, the container of that browsing context is a frame or iframe element, that element has a scrolling attribute, and that attribute's value is an ASCII case-insensitive match for the string "off", "noscrol", or "no", then the user agent is expected to prevent any scrollbars from being shown for the viewport of the Document, 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.
The following rules are also expected to apply, as presentational hints:

In quirks mode, the following rules are also expected to apply:

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.

The dialog element, when its is modal flag is true, is expected to act as if it had a user-agent-level style sheet rule setting the following properties:

- 'position' property to 'fixed'
- 'overflow' property to 'auto'
- 'inset-block-start' property to '0'
- 'inset-block-end' property to '0'
- 'max-width' property to 'calc(100% - 6px - 2em)'
- 'max-height' property to 'calc(100% - 6px - 2em)'

15.3.4 Phrasing content

```
@namespace url(http://www.w3.org/1999/xhtml);

css

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; }
rt { display: ruby-text; }

:link { color: #0000EE; }
:visited { color: #551A8B; }
:link:active, :visited:active { color: #FF0000; }
:link, :visited { text-decoration: underline; cursor: pointer; }

:focus-visible { 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; }

q::before { content: open-quote; }
q::after { content: close-quote; }

br { display-outside: newline; } /* this also has bidi implications */

br { display-outside: break-opportunity; } /* this also has bidi implications */

no { white-space: nowrap; }

```
The following rules are also expected to apply, as presentational hints:\textsuperscript{1183}:

```css
@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; }
```

For the purposes of the CSS ruby model, runs of children of \textit{ruby} elements that are not \textit{rt} or \textit{rp} elements are expected to be wrapped in anonymous boxes whose \texttt{\textit{display} property} has the value \texttt{\textit{ruby-base}}. [CSSRUBY]\textsuperscript{1222}

When a particular part of a ruby has more than one annotation, the annotations should be distributed on both sides of the base text so as to minimize the stacking of ruby annotations on one side.

**Note**

\textit{When it becomes possible to do so, the preceding requirement will be updated to be expressed in terms of CSS ruby. (Currently, CSS ruby does not handle nested \textit{ruby} elements or multiple sequential \textit{rt} elements, which is how this semantic is expressed.)}

User agents that do not support correct ruby rendering are expected to render parentheses around the text of \textit{rt} elements in the absence of \textit{rp} elements.

User agents are expected to support the \texttt{\textit{clear} property} on inline elements (in order to render \textit{br} elements with \texttt{\textit{clear}} attributes) in the manner described in the non-normative note to this effect in CSS.

The initial value for the \texttt{\textit{color} property} is expected to be black. The initial value for the \texttt{\textit{background-color} property} is expected to be 'transparent'. The canvas's background is expected to be white.

When a \textit{font} element has a \texttt{\textit{color} attribute}, its value is expected to be parsed using the \texttt{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\textsuperscript{1183} setting the element's \texttt{\textit{color} property} to the resulting color.

The \textit{font} 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 \texttt{\textit{color} property}.

When a \textit{font} element has a \texttt{\textit{face} attribute}, the user agent is expected to treat the attribute as a presentational hint\textsuperscript{1183} setting the element's \texttt{\textit{font-family} property} to the attribute's value.

When a \textit{font} element has a \texttt{\textit{size} attribute}, the user agent is expected to use the following steps, known as the \texttt{rules for parsing a legacy font size}, to treat the attribute as a presentational hint\textsuperscript{1183} setting the element's \texttt{\textit{font-size} property}:

1. Let \textit{input} be the attribute's value.
2. Let \textit{position} be a pointer into \textit{input}, initially pointing at the start of the string.
3. Skip ASCII whitespace within \textit{input} given \textit{position}.
4. If \textit{position} is past the end of \textit{input}, there is no \texttt{presentational hint\textsuperscript{1183}. Return.}
5. If the character at \textit{position} is a U+002B PLUS SIGN character (+), then let \textit{mode} be \texttt{relative-plus}, and advance \textit{position} to the next character. Otherwise, if the character at \textit{position} is a U+002D HYPHEN-MINUS character (-), then let \textit{mode} be \texttt{relative-minus}, and advance \textit{position} to the next character. Otherwise, let \textit{mode} be \texttt{absolute}.
6. Collect a sequence of code points that are ASCII digits from \textit{input} given \textit{position}, and let the resulting sequence be \textit{digits}.
7. If \textit{digits} is the empty string, there is no \texttt{presentational hint\textsuperscript{1183}. Return.}
8. Interpret \textit{digits} as a base-ten integer. Let \textit{value} be the resulting number.
9. If \textit{mode} is \texttt{relative-plus}, then increment \textit{value} by 3. If \textit{mode} is \texttt{relative-minus}, then let \textit{value} be the result of subtracting \textit{value} from 3.
10. If \textit{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:

<table>
<thead>
<tr>
<th>value</th>
<th><code>font-size</code> keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>'x-small'</td>
</tr>
<tr>
<td>2</td>
<td>'small'</td>
</tr>
<tr>
<td>3</td>
<td>'medium'</td>
</tr>
<tr>
<td>4</td>
<td>'large'</td>
</tr>
<tr>
<td>5</td>
<td>'x-large'</td>
</tr>
<tr>
<td>6</td>
<td>'xx-large'</td>
</tr>
<tr>
<td>7</td>
<td>'xxx-large'</td>
</tr>
</tbody>
</table>

### 15.3.5 Bidirectional text

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 Email 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:

```css
@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, hgroup, nav, section, table, caption, colgroup, col, thead, tbody, tfoot, tr, 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]:is([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 */

```
15.3.6 Sections and headings

The shorthand is used to keep this block at least mildly readable.

```css
@namespace url(http://www.w3.org/1999/xhtml);

article, aside, h1, h2, h3, h4, h5, h6, hgroup, nav, section {
  display: block;
}

h1 { margin-block-start: 0.67em; margin-block-end: 0.67em; font-size: 2.00em; font-weight: bold; }

h2 { margin-block-start: 0.83em; margin-block-end: 0.83em; font-size: 1.50em; font-weight: bold; }

h3 { margin-block-start: 1.00em; margin-block-end: 1.00em; font-size: 1.17em; font-weight: bold; }

h4 { margin-block-start: 1.33em; margin-block-end: 1.33em; font-size: 1.00em; font-weight: bold; }

h5 { margin-block-start: 1.67em; margin-block-end: 1.67em; font-size: 0.83em; font-weight: bold; }

h6 { margin-block-start: 2.33em; margin-block-end: 2.33em; font-size: 0.67em; font-weight: bold; }
```

In the following CSS block, x is shorthand for the following selector: :is(article, aside, nav, section)

```css
@namespace url(http://www.w3.org/1999/xhtml);

x h1 { margin-block-start: 0.83em; margin-block-end: 0.83em; font-size: 1.50em; }

x x h1 { margin-block-start: 1.00em; margin-block-end: 1.00em; font-size: 1.17em; }

x x x h1 { margin-block-start: 1.33em; margin-block-end: 1.33em; font-size: 1.00em; }

x x x x h1 { margin-block-start: 1.67em; margin-block-end: 1.67em; font-size: 0.83em; }

x hgroup > h1 ~ h2 { margin-block-start: 1.00em; margin-block-end: 1.00em; font-size: 1.17em; }

x x hgroup > h1 ~ h2 { margin-block-start: 1.33em; margin-block-end: 1.33em; font-size: 1.00em; }

x x x hgroup > h1 ~ h2 { margin-block-start: 2.33em; margin-block-end: 2.33em; font-size: 0.83em; }

x hgroup > h1 ~ h3 { margin-block-start: 1.33em; margin-block-end: 1.33em; font-size: 1.00em; }

x x hgroup > h1 ~ h3 { margin-block-start: 1.67em; margin-block-end: 1.67em; font-size: 0.83em; }

x x x hgroup > h1 ~ h3 { margin-block-start: 2.33em; margin-block-end: 2.33em; font-size: 0.67em; }

x hgroup > h1 ~ h4 { margin-block-start: 1.67em; margin-block-end: 1.67em; font-size: 0.83em; }

x x hgroup > h1 ~ h4 { margin-block-start: 2.33em; margin-block-end: 2.33em; font-size: 0.67em; }

x hgroup > h1 ~ h5 { margin-block-start: 2.33em; margin-block-end: 2.33em; font-size: 0.67em; }
```

Note

The shorthand is used to keep this block at least mildly readable.

15.3.7 Lists

```css
@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-block-start: 1em; margin-block-end: 1em; }

:is(dir, dl, menu, ol, ul) :is(dir, dl, menu, ol, ul) {
```
The following rules are also expected to apply, as presentational hints:

```css
@namespace url(http://www.w3.org/1999/xhtml);

ol[type="1"], li[type="1"] { list-style-type: decimal; }
ol[type=a s], li[type=a s] { list-style-type: lower-alpha; }
ol[type=A s], li[type=A s] { list-style-type: upper-alpha; }
ol[type=1 s], li[type=1 s] { list-style-type: lower-roman; }
ol[type=I s], li[type=I s] { 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; }
```

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 `li` elements, because CSS doesn't yet provide sufficient hooks for this purpose.

---

15.3.8 Tables

```css
@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 { display: table-cell; }

colgroup[hidden], col[hidden], thead[hidden], tbody[hidden],
tfoot[hidden], tr[hidden] {
  visibility: collapse;
}

table {
```
The following rules are also expected to apply, as presentational hints:

```css
@namespace url(http://www.w3.org/1999/xhtml);

/* CSS 1192 */
```

```
box-sizing: border-box;
border-spacing: 2px;
border-collapse: separate;
text-indent: initial;
}

td, th { padding: 1px; }
th { font-weight: bold; }

caption { text-align: center; }
thead, tbody, tfoot, table > tr { vertical-align: middle; }
tr, td, th { vertical-align: inherit; }
thead, tbody, tfoot, tr { border-color: inherit; }

table[rules=none] > tr > td, table[rules=none] > tr > th,
table[rules=groups] > tr > td, table[rules=groups] > tr > th,
table[rules=rows] > tr > td, table[rules=rows] > tr > th,
table[rules=cols] > tr > td, table[rules=cols] > tr > th,
table[rules=all] > tr > td, table[rules=all] > tr > th,
table[rules=none] > thead > tr > td, table[rules=none] > thead > tr > th,
table[rules=groups] > thead > tr > td, table[rules=groups] > thead > tr > th,
table[rules=rows] > thead > tr > td, table[rules=rows] > thead > tr > th,
table[rules=cols] > thead > tr > td, table[rules=cols] > thead > tr > th,
table[rules=all] > thead > tr > td, table[rules=all] > thead > tr > th,
table[rules=none] > tbody > tr > td, table[rules=none] > tbody > tr > th,
table[rules=groups] > tbody > tr > td, table[rules=groups] > tbody > tr > th,
table[rules=rows] > tbody > tr > td, table[rules=rows] > tbody > tr > th,
table[rules=cols] > tbody > tr > td, table[rules=cols] > tbody > tr > th,
table[rules=all] > tbody > tr > td, table[rules=all] > tbody > tr > th,
table[rules=none] > tfoot > tr > td, table[rules=none] > tfoot > tr > th,
table[rules=groups] > tfoot > tr > td, table[rules=groups] > tfoot > tr > th,
table[rules=rows] > tfoot > tr > td, table[rules=rows] > tfoot > tr > th,
table[rules=cols] > tfoot > tr > td, table[rules=cols] > tfoot > tr > th,
table[rules=all] > tfoot > tr > td, table[rules=all] > tfoot > tr > th {
border-color: black;
}

caption[align=bottom] { caption-side: bottom; }
p[align=left], h1[align=left], h2[align=left], h3[align=left], h4[align=left], h5[align=left], h6[align=left] {
  text-align: left;
}
p[align=right], h1[align=right], h2[align=right], h3[align=right], h4[align=right], h5[align=right], h6[align=right] {
  text-align: right;
}
```
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:

```css
@namespace url(http://www.w3.org/1999/xhtml);

:is(table, thead, tbody, tfoot, tr) > form { display: none !important; }
```

The table element's cellspacing attribute maps to the pixel length property \( p1184 \) 'border-spacing' on the element.

The table element's cellpadding attribute maps to the pixel length properties \( p1184 \) '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 height attribute maps to the dimension property (ignoring zero) \( p1184 \) 'height' on the table element.

The col element's width attribute maps to the dimension property (ignoring zero) \( p1184 \) 'width' on the table element.

The thead, tbody, and tfoot elements' height attribute maps to the dimension property \( p1184 \) 'height' on the element.

The tr element's height attribute maps to the dimension property (ignoring zero) \( p1184 \) 'height' on the element.

The td and th elements' height attributes map to the dimension property (ignoring zero) \( p1184 \) 'height' on the element.

The td and th elements' width attributes map to the dimension property (ignoring zero) \( p1184 \) 'width' on the element.

The thead, tbody, tfoot, tr, and th elements, when they have an align attribute whose value is an ASCII case-insensitive match for 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 \( p1183 \), and to align descendants \( p1184 \) to the center.

The 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 \( p1183 \), and to align descendants \( p1184 \) to the left.

The 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 \( p1183 \), and to align descendants \( p1184 \) to the right.

User agents are expected to have a rule in their user agent style sheet 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 \( p1183 \) 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 \( p1183 \) 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 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 nonzero 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'.

### 15.3.9 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`, `dir`, `dl`, `h1`, `h2`, `h3`, `h4`, `h5`, `h6`, `listing`, `menu`, `ol`, `p`, `plaintext`, `pre`, `ul`, `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-block-start` 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-block-end` 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-block-start` 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-block-end` property to zero.

### 15.3.10 Form controls

```css
@namespace url(http://www.w3.org/1999/xhtml);

input, select, button, textarea {
  letter-spacing: initial;
  word-spacing: initial;
  line-height: initial;
  text-transform: initial;
  text-indent: initial;
  text-shadow: initial;
}

input, select, textarea {
  text-align: initial;
}
```
In quirks mode, the following rules are also expected to apply:

```
@media (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 described in the Widgets section, which describes the look and feel of the control.

### 15.3.11 The hr element

```
@media (namespace url(http://www.w3.org/1999/xhtml));
hr {
  color: gray;
  border-style: inset;
  border-width: 1px;
  margin-block-start: 0.5em;
  margin-inline-end: auto;
  margin-block-end: 0.5em;
  margin-inline-start: auto;
  overflow: hidden;
}
```

The following rules are also expected to apply, as presentational hints.

```
@media (namespace url(http://www.w3.org/1999/xhtml));
hr[align=left i] { margin-left: 0; margin-right: auto; }
hr[align=right i] { margin-left: auto; margin-right: 0; }
hr[align=center i] { 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 parsing 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.

15.3.12 The `fieldset` and `legend` elements

The `fieldset` element, when it generates a CSS box, is expected to act as follows:

- The element is expected to establish a new block formatting context.
- The `'display'` property is expected to act as follows:
  - If the computed value of `'display'` is a value such that the outer display type is 'inline', then behave as 'inline-block'.
  - Otherwise, behave as 'flow-root'.

  **Note**
  This does not change the computed value.

- If the element's box has a child box that matches the conditions in the list below, then the first such child box is the 'fieldset' element's rendered legend:
  - The child is a `legend` element.
  - The child's used value of `'float'` is 'none'.
  - The child's used value of `'position'` is not 'absolute' or 'fixed'.

- If the element has a rendered legend, then the border is expected to not be painted behind the rectangle defined as follows, using the writing mode of the fieldset:

  1. The block-start edge of the rectangle is the smaller of the block-start edge of the rendered legend's margin...
rectangle at its static position (ignoring transforms), and the block-start outer edge of the fieldset’s border.

2. The block-end edge of the rectangle is the larger of the block-end edge of the rendered legend’s margin rectangle at its static position (ignoring transforms), and the block-end outer edge of the fieldset’s border.

3. The inline-start edge of the rectangle is the smaller of the inline-start edge of the rendered legend’s border rectangle at its static position (ignoring transforms), and the inline-start outer edge of the fieldset’s border.

4. The inline-end edge of the rectangle is the larger of the inline-end edge of the rendered legend’s border rectangle at its static position (ignoring transforms), and the inline-end outer edge of the fieldset’s border.

   • The space allocated for the element’s border on the block-start side is expected to be the element’s ‘border-block-start-width’ or the rendered legend’s margin box size in the fieldset’s block-flow direction, whichever is greater.

   • For the purpose of calculating the used ‘block-size’, if the computed ‘block-size’ is not ‘auto’, the space allocated for the rendered legend’s margin box that spills out past the border, if any, is expected to be substracted from the ‘block-size’. If the content box’s block-size would be negative, then let the content box’s block-size be zero instead.

   • If the element has a rendered legend, then that element is expected to be the first child box.

   • The anonymous fieldset content box is expected to appear after the rendered legend and is expected to contain the content (including the ‘::before’ and ‘::after’ pseudo-elements) of the fieldset element except for the rendered legend, if there is one.

   • The used value of the ‘padding-top’, ‘padding-right’, ‘padding-bottom’, and ‘padding-left’ properties are expected to be zero.

   • For the purpose of calculating the min-content inline size, use the greater of the min-content inline size of the rendered legend and the min-content inline size of the anonymous fieldset content box.

   • For the purpose of calculating the max-content inline size, use the greater of the max-content inline size of the rendered legend and the max-content inline size of the anonymous fieldset content box.

A fieldset element’s rendered legend, if any, is expected to act as follows:

   • The element is expected to establish a new formatting context for its contents. The type of this formatting context is determined by its ‘display’ value, as usual.

   • The ‘display’ property is expected to behave as if its computed value was blockified.

   • If the computed value of ‘inline-size’ is ‘auto’, then the used value is the fit-content inline size.

   • The element is expected to be positioned in the inline direction as is normal for blocks (e.g., taking into account margins and the ‘justify-self’ property).

   • The element’s box is expected to be constrained in the inline direction by the inline content size of the fieldset as if it had used its computed inline padding.

   • The element is expected to be positioned in the block-flow direction such that its border box is centered over the border on the block-start side of the fieldset element.

A fieldset element’s anonymous fieldset content box is expected to act as follows:

   • The ‘display’ property is expected to act as follows:
      
      ◦ If the computed value of ‘display’ on the fieldset element is ‘grid’ or ‘inline-grid’, then set the used value to ‘grid’.
      
      ◦ If the computed value of ‘display’ on the fieldset element is ‘flex’ or ‘inline-flex’, then set the used value to ‘flex’.

   • For example, if the fieldset has a specified padding of 50px, then the rendered legend will be positioned 50px in from the fieldset’s border. The padding will further apply to the anonymous fieldset content box instead of the fieldset element itself.

   • The element is expected to be positioned in the block-flow direction such that its border box is centered over the border on the block-start side of the fieldset element.
Otherwise, set the used value to 'flow-root'.

- The following properties are expected to inherit from the `fieldset` element:
  - 'align-content'
  - 'align-items'
  - 'border-radius'
  - 'column-count'
  - 'column-fill'
  - 'column-gap'
  - 'column-rule'
  - 'column-width'
  - 'flex-direction'
  - 'flex-wrap'
  - 'grid-auto-columns'
  - 'grid-auto-rows'
  - 'grid-column-gap'
  - 'grid-row-gap'
  - 'grid-template-areas'
  - 'grid-template-columns'
  - 'grid-template-rows'
  - 'justify-content'
  - 'justify-items'
  - 'overflow'
  - 'padding-bottom'
  - 'padding-left'
  - 'padding-right'
  - 'padding-top'
  - 'text-overflow'
  - 'unicode-bidi'

- The 'block-size' property is expected to be set to '100%'.

- For the purpose of calculating percentage padding, act as if the padding was calculated for the `fieldset` element.

**Note**

15.4 Replaced elements

The following elements can be replaced elements: `audio`, `canvas`, `embed`, `iframe`, `img`, `input`, `object`, and
15.4.1 Embedded content

The `embed`, `iframe`, and `video` elements are expected to be treated as replaced elements.

A `canvas` element that represents an image, plugin, or its nested browsing context is expected to be treated as a replaced 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 its nested browsing context is expected to be treated as a replaced element. Other `object` 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 overlayed 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:

```css
@namespace url(http://www.w3.org/1999/xhtml);

iframe { border: 2px inset; }
video { object-fit: contain; }
```

15.4.2 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 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, and the element already has intrinsic dimensions (e.g., from the dimension attributes or CSS rules)

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 the non-replaced phrasing element whose content is the text, optionally 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 a replaced element whose intrinsic dimensions are 0. (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 UA 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:

```css
@namespace url(http://www.w3.org/1999/xhtml);

img[align=left] { margin-right: 3px; }
img[align=right] { margin-left: 3px; }
```

15.4.3 Attributes for embedded content and images

The following CSS rules are expected to apply as presentational hints:

```css
@namespace url(http://www.w3.org/1999/xhtml);

iframe[frameborder='0'], iframe[frameborder=no] { border: none; }
embed[align=left], iframe[align=left], img[align=left],
input[type=image][align=left], object[align=left] { float: left;
```
When an `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 `embed`, `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 `embed`, `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 `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.

The intrinsic aspect ratio for an `img` element is computed as follows:

1. If `img`'s `current request` is `available`, and has an intrinsic aspect ratio, then use that intrinsic aspect ratio.
2. If `img`'s `width` and `height` attribute values, when parsed using the rules for parsing dimension values, are both not an error, not a percentage, and non-zero, then use the ratio resulting from dividing the `width` attribute value by the `height` attribute value.
3. Otherwise, `img` has no intrinsic aspect ratio.
15.4.4 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 img or object element.

For the purposes of the rendering, only the 'cursor' property is expected to have any effect on the shape.

Example

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

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.

15.5 Widgets

15.5.1 Introduction

The elements defined in this section can be rendered in a variety of manners, within the guidelines provided below. User agents are encouraged to 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.

15.5.2 Button layout

Button layout is as follows:

- The 'display' property is expected to act as follows:
  - If the computed value of 'display' is 'inline-grid', 'grid', 'inline-flex', or 'flex', then behave as the computed value.
  - Otherwise, if the computed value of 'display' is a value such that the outer display type is 'inline', then behave as 'inline-block'.
  - Otherwise, behave as 'flow-root'.
- The element is expected to establish a new formatting context for its contents. The type of this formatting context is determined by its 'display' value, as usual.
- If the element is absolutely-positioned, then for the purpose of the CSS visual formatting model, act as if the element is a replaced element.
- If the computed value of 'inline-size' is 'auto', then the used value is the fit-content inline size.
- For the purpose of the 'normal' keyword of the 'align-self' property, act as if the element is a replaced element.
- If the element is an input element, or if it is a button element and its computed value for 'display' is not 'inline-grid', 'grid', 'inline-flex', or 'flex', then the element's box has a child anonymous button content box with the following behaviors:
  - The box is a block-level block container that establishes a new block formatting context (i.e., 'display' is 'flow-root').
  - If the box does not overflow in the horizontal axis, then it is centered horizontally.
  - If the box does not overflow in the vertical axis, then it is centered vertically.

Otherwise, there is no anonymous button content box.
15.5.3 The **button** element

The **button** element, when it generates a **CSS box**, is expected to depict a button and to use **button layout** whose **anonymous button content box**'s contents (if there is an **anonymous button content box**) are the child boxes the element's box would otherwise have.

15.5.4 The **details** and **summary** elements

```css
@namespace url(http://www.w3.org/1999/xhtml);

summary {
  display: list-item;
  counter-increment: list-item 0;
  list-style: disclosure-closed inside;
}
details[open] > summary {
  list-style-type: disclosure-open;
}
```

The **details** element is expected to render as a **block box**. The element is also expected to have an internal **shadow tree** with two **slots**. The first **slot** is expected to take the **details** element's first **summary** element child, if any. The second **slot** is expected to take the **details** element's remaining descendants, if any.

The **details** element's first **summary** element child, if any, is expected to allow the user to request the details be shown or hidden.

The **details** element's second **slot** is expected to be removed from the rendering when the **details** element does not have an **open** attribute.

15.5.5 The **input** element as a text entry widget

An **input** element whose **type** attribute is in the **Text**, **Search**, **Telephone**, **URL**, or **Email** state, is expected to render as an **'inline-block'** box depicting a text control. Additionally, 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.

An **input** element whose **type** attribute is in the **Password** state is expected to render as an **'inline-block'** box depicting a text control that obscures data entry.

If these text controls provide a text selection, then, when the user changes the current selection, the user agent is expected to queue an element task on the user interaction task source given the **input** element to fire an event named **select** at the element, with the **bubbles** attribute initialized to true.

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 \((size-1) \times \text{avg} + \text{max}\), 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.)
15.5.6 The input element as domain-specific widgets §06

An input element whose type attribute is in the Date state is expected to render as an 'inline-block' box depicting a date control.

An input element whose type attribute is in the Month state is expected to render as an 'inline-block' box depicting a month control.

An input element whose type attribute is in the Week state is expected to render as an 'inline-block' box depicting a week control.

An input element whose type attribute is in the Time state is expected to render as an 'inline-block' box depicting a time control.

An input element whose type attribute is in the Local Date and Time state is expected to render as an 'inline-block' box depicting a local date and time control.

An input element whose type attribute is in the Number state 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.

15.5.7 The input element as a range control §06

An input element whose type attribute is in the Range state 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.

15.5.8 The input element as a color well §06

An input element whose type attribute is in the Color state is expected to depict 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. The element, when it generates a CSS box, is expected to use button layout, that has no child boxes of the anonymous button content box. The anonymous button content box is expected to have a presentational hint setting the 'background-color' property to the element's value.

Predefined suggested values (provided by the list attribute) are expected to be shown in the color picker interface, not on the color well itself.

15.5.9 The input element as a checkbox and radio button widgets §06

An input element whose type attribute is in the Checkbox state is expected to render as an 'inline-block' box containing a single checkbox control, with no label.

An input element whose type attribute is in the Radio Button state is expected to render as an 'inline-block' box containing a single radio button control, with no label.
15.5.10 The `input` element as a file upload control

An `input` element whose `type` attribute is in the `File Upload` state, when it generates a CSS box, 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. The button is expected to use `button layout` and match the `:file-selector-button` pseudo-element. The contents of its anonymous button content box are expected to be `implementation-defined` (and possibly locale-specific) text, for example “Choose file”.

15.5.11 The `input` element as a button

An `input` element whose `type` attribute is in the `Submit Button`, `Reset Button`, or `Button` state, when it generates a CSS box, is expected to depict a button and use `button layout` and the contents of the anonymous button content box are expected to be the text of the element's `value` attribute, if any, or text derived from the element's `type` attribute in an `implementation-defined` (and probably locale-specific) fashion, if not.

15.5.12 The `marquee` element

CSS

```
@namespace url(http://www.w3.org/1999/xhtml);

marquee {
  display: inline-block;
  text-align: initial;
}
```

The `marquee` element, while `turned on`, 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

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

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**

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:

<table>
<thead>
<tr>
<th><code>direction</code> attribute state</th>
<th>Direction of animation</th>
<th>Start edge</th>
<th>End edge</th>
<th>Opposite direction</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>left</code></td>
<td>← Right to left</td>
<td>Right</td>
<td>Left</td>
<td>→ Left to Right</td>
</tr>
<tr>
<td><code>right</code></td>
<td>→ Left to Right</td>
<td>Left</td>
<td>Right</td>
<td>← Right to left</td>
</tr>
<tr>
<td><code>up</code></td>
<td>↑ Up (Bottom to Top)</td>
<td>Bottom</td>
<td>Top</td>
<td>↓ Down (Top to Bottom)</td>
</tr>
<tr>
<td><code>down</code></td>
<td>↓ Down (Top to Bottom)</td>
<td>Top</td>
<td>Bottom</td>
<td>↑ Up (Bottom to Top)</td>
</tr>
</tbody>
</table>

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 `width` attribute of a `marquee` 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.

### 15.5.13 The `meter` element

The `meter` element is expected to render as an `inline-block` box with a `height` of '1em' and a `width` of '5em', 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.

### 15.5.14 The `progress` element

The `progress` 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

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.

15.5.15 The select element

A select element is either a list box or a drop-down box, depending on its attributes.

A select element whose multiple attribute is present is expected to render as a multi-select list box.

A select element whose multiple attribute is absent, and whose display size is greater than 1, 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 element's labels plus the width of a scrollbar.

A select element whose multiple attribute is absent, and whose display size is 1, is expected to render as a one-line drop-down box whose width is the width of the select element'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.
15.5.16 The `textarea` element

The `textarea` element is expected to render as an `inline-block` box depicting a multiline text control. If this multiline text control provides a selection, then, when the user changes the current selection, the user agent is expected to queue an element task on the user interaction task source given the `textarea` element to fire an event named `select` at the element, with the `bubbles` attribute initialized to true.

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×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 scrollbar, 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'.

15.6 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 `wxh` 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
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. 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 its nested browsing context in this case, and its nested browsing context is expected to have a viewport with zero width and zero height.

### 15.7 Interactive media

#### 15.7.1 Links, forms, and navigation

User agents are expected to allow the user to control aspects of hyperlink activation and 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 are expected to inform the user of whether a hyperlink includes hyperlink auditing, and to let them know at a minimum which domains will be contacted as part of such auditing.

User agents may allow users to navigate to the URLs indicated by the cite attributes on cite, blockquote, ins, and del elements.

User agents may surface created by link elements in their user interface, as discussed previously.

#### 15.7.2 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, or by showing the advisory information of the elements under the user’s finger on a touch device as the user pans around the screen.

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 nonzero 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**

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**

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.

### 15.7.3 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.

### 15.7.4 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, popup 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**

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:

```html
<p dir="rtl" lang="he">
  <label>
    בחר שפת תכנות:
  </label>
  <select>
    <option dir="ltr">C++</option>
    <option dir="ltr">C#</option>
    <option dir="ltr">FreePascal</option>
    <option dir="ltr">F#</option>
  </select>
</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.
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.

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.

For a more complex example, consider the following script:

```javascript
/* 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 "أفهم لا", 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: "أفهم لا, and Wilma will get the car, AGAIN! Ok, Fred! أفهم لا"

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.);
}

15.8 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.

15.9 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 style 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.
16 Obsolete features

16.1 Obsolete but conforming features

Features listed in this section will trigger warnings in conformance checkers.

Authors should not specify a `border` attribute on an `img` element. If the attribute is present, its value must be the string "0". CSS should be used instead.

Authors should not specify a `charset` attribute on a `script` element. If the attribute is present, its value must be an ASCII case-insensitive match for "utf-8". (This has no effect in a document that conforms to the requirements elsewhere in this standard of being encoded as UTF-8.)

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 a value for the `type` attribute on `script` elements that is the empty string or a JavaScript MIME type essence match. Instead, they should omit the attribute, which has the same effect.

Authors should not specify a `type` attribute on a `style` element. If the attribute is present, its value must be an ASCII case-insensitive match for "text/css".

Authors should not specify the `name` attribute on an `a` element. 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 tree 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 tree. 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 fragments 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 control with a useful width.

16.1.1 Warnings for obsolete but conforming features

To ease the transition from HTML4 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 a `border` attribute on an `img` element if its value is the string "0".
- The presence of a `charset` attribute on a `script` element if its value is an ASCII case-insensitive match for "utf-8".
- 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 `type` attribute on a `script` element if its value is a JavaScript MIME type essence match.
- The presence of a `type` attribute on a `style` element if its value is an ASCII case-insensitive match for "text/css".
- 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**

For example, a validator could report some pages as "Valid HTML" and others as "Valid HTML with warnings".

## 16.2 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 `ul` instead.

- **frame**
  - **frameset**
    - 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 control` combination instead.

- **keygen**
  - For enterprise device management use cases, use native on-device management capabilities.

  For certificate enrollment use cases, use the Web Cryptography API to generate a keypair for the certificate, and then export the certificate and key to allow the user to install them manually. [WEBCRYPTO]

- **listing**
  - Use `pre` and `code` instead.

- **menuitem**
  - To implement a custom context menu, use script to handle the `contextmenu` event.

- **nextid**
  - Use GUIDs instead.

- **noembed**
  - Use `object` instead of `embed` when fallback is necessary.

- **plaintext**
  - Use the "text/plain" MIME type instead.

- **rb**

- **rtc**
  - Providing the ruby base directly inside the `ruby` element or using nested `ruby` elements is sufficient.
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 `&lt;` and `&amp;` respectively.
</xmp>

basefont
big
blink
center
tt
<font>
<marquee>
multicol
nobr
spacer

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 `strong` 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.

<charset on `script` elements (except as noted in the previous section)>
Omit the attribute. Both documents and scripts are required to use UTF-8, so it is redundant to specify it on the `script` element since it inherits from the document.

<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 `img` elements>
<name on `option` elements>
Use the `id` attribute instead.

<rev on `a` elements>
<rev on `link` elements>
Use the `rel` attribute instead, with an opposite term. (For example, instead of `rev="made"`, use `rel="author"`.)
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.

hreflang on area elements
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
    Unnecessary. Omit it altogether.

manifest on html elements
    Use service workers instead. [SW]

version on html elements
    Unnecessary. Omit it altogether.

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.

longdesc on iframe elements
longdesc on img elements
    Use a regular a element to link to the description, or (in the case of images) use an image map to provide a link from the image to the image's description.

lowsrc on img 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.

type on menu elements
    To implement a custom context menu, use script to handle the contextmenu event. For toolbar menus, omit the attribute.

label on menu elements
contextmenu on all elements
onshow on all elements
    To implement a custom context menu, use script to handle the contextmenu event.

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\textsuperscript{p373} elements

Repeat the object\textsuperscript{p373} element completely each time the resource is to be reused.

Standby on object\textsuperscript{p373} elements

Optimize the linked resource so that it loads quickly or, at least, incrementally.

typemustmatch on object\textsuperscript{p373} elements

Avoid using object\textsuperscript{p373} elements with untrusted resources.

type on param\textsuperscript{p379} elements

value on param\textsuperscript{p379} elements

Use the name\textsuperscript{p379} and value\textsuperscript{p379} attributes without declaring value types.

Language on script\textsuperscript{p613} elements (except as noted in the previous section)

Omit the attribute for JavaScript; for data blocks\textsuperscript{p615}, use the type\textsuperscript{p614} attribute instead.

Event on script\textsuperscript{p613} elements

For on script\textsuperscript{p613} elements

Use DOM events mechanisms to register event listeners. [DOM]\textsuperscript{p1272}

type on style\textsuperscript{p176} elements (except as noted in the previous section)

Omit the attribute for CSS; for data blocks\textsuperscript{p615}, use script\textsuperscript{p613} as the container instead of style\textsuperscript{p176}.

datapagesize on table\textsuperscript{p450} elements

Unnecessary. Omit it altogether.

Summary on table\textsuperscript{p450} elements

Use one of the techniques for describing tables\textsuperscript{p455} given in the table\textsuperscript{p450} section instead.

Abbr on td\textsuperscript{p466} elements

Use text that begins in an unambiguous and terse manner, and include any more elaborate text after that. The title\textsuperscript{p140} attribute 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\textsuperscript{p467} (which has an abbr\textsuperscript{p468} attribute).

Axis on td\textsuperscript{p466} and th\textsuperscript{p467} elements

Use the scope\textsuperscript{p467} attribute on the relevant th\textsuperscript{p467}.

Scope on td\textsuperscript{p466} elements

Use th\textsuperscript{p467} elements for heading cells.

datasrc on a\textsuperscript{p239}, button\textsuperscript{p535}, div\textsuperscript{p238}, iframe\textsuperscript{p1224}, img\textsuperscript{p321}, input\textsuperscript{p493}, label\textsuperscript{p499}, legend\textsuperscript{p565}, marquee\textsuperscript{p1222}, object\textsuperscript{p373}, option\textsuperscript{p494}, select\textsuperscript{p537}, span\textsuperscript{p280}, table\textsuperscript{p450}, and textarea\textsuperscript{p548} elements

datafld on a\textsuperscript{p239}, button\textsuperscript{p535}, div\textsuperscript{p238}, fieldset\textsuperscript{p562}, frame\textsuperscript{p1224}, iframe\textsuperscript{p362}, img\textsuperscript{p321}, input\textsuperscript{p493}, label\textsuperscript{p499}, legend\textsuperscript{p565}, marquee\textsuperscript{p1222}, object\textsuperscript{p373}, param\textsuperscript{p379}, select\textsuperscript{p537}, span\textsuperscript{p280}, and textarea\textsuperscript{p548} elements

dataformats on button\textsuperscript{p535}, div\textsuperscript{p238}, input\textsuperscript{p493}, label\textsuperscript{p499}, legend\textsuperscript{p565}, marquee\textsuperscript{p1222}, object\textsuperscript{p373}, option\textsuperscript{p494}, select\textsuperscript{p537}, span\textsuperscript{p280}, and table\textsuperscript{p450} elements

Use script and a mechanism such as XMLHttpRequest to populate the page dynamically. [XHR]\textsuperscript{p1278}

dropzone on all elements

Use script to handle the dragenter\textsuperscript{p811} and dragover\textsuperscript{p811} events instead.
The marquee element is a presentational element that animates content. CSS transitions and animations are a more appropriate mechanism. [CSSANIMATIONS][CSSTRANSITIONS]

The marquee element must implement the HTMLMarqueeElement interface.

```idl
[Exposed=Window]
interface HTMLMarqueeElement : HTMLElement {
    [HTMLConstructor] constructor();

    [CEReactions] attribute DOMString behavior;
    [CEReactions] attribute DOMString bgColor;
    [CEReactions] attribute DOMString direction;
    [CEReactions] attribute DOMString nowrap;
    [CEReactions] attribute DOMString valingAlign;  
    [CEReactions] attribute DOMString width;
}
```
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.

The `behavior` content attribute on marquee elements is an enumerated attribute with the following keywords (all non-conforming):

<table>
<thead>
<tr>
<th>Keyword</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>scroll</td>
<td>scroll</td>
</tr>
<tr>
<td>slide</td>
<td>slide</td>
</tr>
<tr>
<td>alternate</td>
<td>alternate</td>
</tr>
</tbody>
</table>

The missing value default and invalid value default are the scroll state.

The `direction` content attribute on marquee elements is an enumerated attribute with the following keywords (all non-conforming):

<table>
<thead>
<tr>
<th>Keyword</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>left</td>
</tr>
<tr>
<td>right</td>
<td>right</td>
</tr>
<tr>
<td>up</td>
<td>up</td>
</tr>
<tr>
<td>down</td>
<td>down</td>
</tr>
</tbody>
</table>

The missing value default and invalid value default are 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` 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` 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` 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 return.
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.

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.

16.3.2 Frames

The `frameset` element acts as the `body element` in documents that use frames.

The `frameset` element must implement the `HTMLFrameSetElement` interface.

```idl
[Exposed=Window]
interface HTMLFrameSetElement : HTMLElement {
    [HTMLConstructor] constructor();

    [CEReactions] attribute DOMString cols;
    [CEReactions] attribute DOMString rows;
};
```

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 `event handlers` of the `Window` object named by the `Window-reflecting body element event handler set`, exposed on the `frameset` element, replace the generic `event handlers` with the same names normally supported by `HTML elements`.

The `frame` element has 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 run these steps:

1. Create a new nested browsing context for element.
2. Process the frame attributes\(^{1225}\) for element, with initialInsertion\(^{1225}\) set to true.

When a frame\(^{1224}\) element stops being an active frame element\(^{1224}\), the user agent must discard\(^{834}\) the element's nested browsing context\(^{1216}\), and then set the element's nested browsing context\(^{1216}\) to null.

Whenever a frame\(^{1224}\) element with a non-null nested browsing context\(^{1216}\) has its src attribute set, changed, or removed, the user agent must process the frame attributes\(^{1225}\).

To process the frame attributes for an element element, with an optional boolean initialInsertion:

1. If element has a src attribute specified, or initialInsertion is false, then run the shared attribute processing steps for iframe and frame elements\(^{364}\) given element.

The frame\(^{1224}\) element potentially delays the load event\(^{365}\).

The frame\(^{1224}\) element must implement the HTMLFrameElement\(^{1225}\) interface.

```idl
[Exposed=Window]
interface HTMLFrameElement : HTMLElement {
    [HTMLConstructor] constructor();

    [CEReactions] attribute DOMString name;
    [CEReactions] attribute DOMString scrolling;
    [CEReactions] attribute USVString src;
    [CEReactions] attribute DOMString frameBorder;
    [CEReactions] attribute USVString longDesc;
    [CEReactions] attribute boolean noResize;
    readonly attribute Document? contentDocument;
    readonly attribute WindowProxy? contentWindow;

    [CEReactions] attribute [LegacyNullToEmptyString] DOMString marginHeight;
    [CEReactions] attribute [LegacyNullToEmptyString] DOMString marginWidth;
};
```

The name, scrolling, and src IDL attributes of the frame\(^{1224}\) element must reflect\(^{95}\) the respective content attributes of the same name. For the purposes of reflection, the frame\(^{1224}\) element's src content attribute is defined as containing a URL.

The frameBorder IDL attribute of the frame\(^{1224}\) element must reflect\(^{95}\) the element's frameborder content attribute.

The longDesc IDL attribute of the frame\(^{1224}\) element must reflect\(^{95}\) the element's longdesc content attribute, which for the purposes of reflection is defined as containing a URL.

The noResize IDL attribute of the frame\(^{1224}\) element must reflect\(^{95}\) the element's noresize content attribute.

The contentDocument IDL attribute, on getting, must return the frame\(^{1224}\) element's content document\(^{818}\).

The contentWindow IDL attribute must return the WindowProxy\(^{837}\) object of the frame\(^{1224}\) element's nested browsing context\(^{1216}\), if the element's nested browsing context\(^{1216}\) is non-null, or return null otherwise.

The marginHeight IDL attribute of the frame\(^{1224}\) element must reflect\(^{95}\) the element's marginheight content attribute.

The marginWidth IDL attribute of the frame\(^{1224}\) element must reflect\(^{95}\) the element's marginwidth content attribute.

16.3.3 Other elements, attributes and APIs \(^{5}^{12}\) \(^{25}\)

User agents must treat acronym\(^{1217}\) elements in a manner equivalent to abbr\(^{2051}\) elements in terms of semantics and for purposes of rendering.

```idl
partial interface HTMLAnchorElement {
```
The `coords`, `charset`, `name`, `rev`, and `shape` IDL attributes of the `a` element must reflect the respective content attributes of the same name.

```idl
partial interface HTMLAreaElement {
  [CEReactions] attribute DOMString coords;
  [CEReactions] attribute DOMString charset;
  [CEReactions] attribute DOMString name;
  [CEReactions] attribute DOMString rev;
  [CEReactions] attribute DOMString shape;
};
```

The `noHref` IDL attribute of the `area` element must reflect the element's `nohref` content attribute.

```idl
partial interface HTMLBodyElement {
  [CEReactions] attribute [LegacyNullToEmptyString] DOMString text;
  [CEReactions] attribute [LegacyNullToEmptyString] DOMString link;
  [CEReactions] attribute [LegacyNullToEmptyString] DOMString vLink;
  [CEReactions] attribute [LegacyNullToEmptyString] DOMString aLink;
  [CEReactions] attribute DOMString bgColor;
  [CEReactions] 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` content is not defined to contain a URL, despite rules regarding its handling in the rendering section above.)

```idl
partial interface HTMLBRElement {
  [CEReactions] attribute DOMString clear;
};
```

The `clear` IDL attribute of the `br` element must reflect the content attribute of the same name.

```idl
partial interface HTMLTableCaptionElement {
  [CEReactions] attribute DOMString align;
};
```

The `align` IDL attribute of the `caption` element must reflect the content attribute of the same name.
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.

```idl
[Exposed=Window]
interface HTMLDirectoryElement : HTMLElement {
    [HTMLConstructor] constructor();
    [CEReactions] attribute boolean compact;
};
```

The `compact` IDL attribute of the `dir` element must `reflect` the content attribute of the same name.

```idl
partial interface HTMLDivElement {
    [CEReactions] attribute DOMString align;
};
```

The `align` IDL attribute of the `div` element must `reflect` the content attribute of the same name.

```idl
partial interface HTMLListElement {
    [CEReactions] attribute boolean compact;
};
```

The `compact` IDL attribute of the `dl` element must `reflect` the content attribute of the same name.

```idl
partial interface HTMLEmbedElement {
    [CEReactions] attribute DOMString align;
    [CEReactions] 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.

```idl
[Exposed=Window]
interface HTMLFontElement : HTMLElement {
    [HTMLConstructor] constructor();
    [CEReactions] attribute [LegacyNullToEmptyString] DOMString color;
    [CEReactions] attribute DOMString face;
};
```
The color, face, and size IDL attributes of the font element must reflect the respective content attributes of the same name.

```
partial interface HTMLHeadingElement {
  [CEReactions] 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 DOM.)

```
partial interface HTMLHRElement {
  [CEReactions] attribute DOMString align;
  [CEReactions] attribute DOMString color;
  [CEReactions] attribute boolean noShade;
  [CEReactions] attribute DOMString size;
  [CEReactions] 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 {
  [CEReactions] attribute DOMString version;
};
```

The version IDL attribute of the html element must reflect the content attribute of the same name.

```
partial interface HTMLIFrameElement {
  [CEReactions] attribute DOMString align;
  [CEReactions] attribute DOMString scrolling;
  [CEReactions] attribute DOMString frameborder;
  [CEReactions] attribute USVString longDesc;
  [CEReactions] attribute [LegacyNullToEmptyString] DOMString marginHeight;
  [CEReactions] attribute [LegacyNullToEmptyString] 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 longDesc IDL attribute of the iframe element must reflect the element’s longdesc content attribute, which for the purposes of reflection is defined as containing a URL.

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 {
  [CEReactions] attribute DOMString name;
  [CEReactions] attribute USVString lowsrc;
  [CEReactions] attribute DOMString align;
  [CEReactions] attribute unsigned long hspace;
  [CEReactions] attribute unsigned long vspace;
  [CEReactions] attribute USVString longDesc;
  [CEReactions] attribute [LegacyNullToEmptyString] 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 `longDesc` IDL attribute of the `img` element must reflect the element's `longdesc` content attribute, which for the purposes of reflection is defined as containing a URL.

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 {
  [CEReactions] attribute DOMString align;
  [CEReactions] 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 {
  [CEReactions] attribute DOMString align;
};
```

The `align` IDL attribute of the `legend` element must reflect the content attribute of the same name.

```
partial interface HTMLLIElement {
  [CEReactions] attribute DOMString type;
};
```

The `type` IDL attribute of the `li` element must reflect the content attribute of the same name.

```
partial interface HTMLLinkElement {
  [CEReactions] attribute DOMString charset;
  [CEReactions] attribute DOMString rev;
  [CEReactions] attribute DOMString target;
};
```

The `charset`, `rev`, 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
The `compact` IDL attribute of the `menu` element must reflect the content attribute of the same name.

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.

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.

```idl
partial interface HTMLListElement {
    [CEReactions] attribute boolean compact;
}
```

The `compact` IDL attribute of the `ol` element must reflect the content attribute of the same name.

```idl
partial interface HTMLParagraphElement {
    [CEReactions] attribute DOMString align;
}
```

The `align` IDL attribute of the `p` element must reflect the content attribute of the same name.

```idl
partial interface HTMLParamElement {
    [CEReactions] attribute DOMString type;
    [CEReactions] 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.)

```idl
partial interface HTMLPreElement {
    [CEReactions] attribute long width;
}
```

The `width` IDL attribute of the `pre` element must reflect the content attribute of the same name.

```idl
partial interface HTMLStyleElement {
    [CEReactions] attribute DOMString type;
}
```

The `type` IDL attribute of the `style` element must reflect the element's `type` content attribute.

```idl
partial interface HTMLScriptElement {
    [CEReactions] attribute DOMString charset;
    [CEReactions] attribute DOMString event;
    [CEReactions] attribute DOMString htmlFor;
}
```

The `charset` and `event` IDL attributes of the `script` element must reflect the respective content attributes of the same name.

The `htmlFor` IDL attribute of the `script` element must reflect the element's `for` content attribute.

```idl
partial interface HTMLTableElement {
```
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.

The `align`, `axis`, `height`, and `width` IDL attributes of the `td` and `th` elements must reflect the content attribute 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 `vAlign` IDL attribute of the `td` and `th` element must reflect the elements' `valign` 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` elements must reflect the elements' `valign` content attributes.

The *bgColor* IDL attribute of the `td` and `th` elements must reflect the elements' `bgcolor` content attributes.

```idl
partial interface HTMLTableRowElement {
  [CEReactions] attribute DOMString align;
  [CEReactions] attribute DOMString ch;
  [CEReactions] attribute DOMString chOff;
  [CEReactions] attribute DOMString vAlign;
  [CEReactions] attribute [LegacyNullToEmptyString] 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.

```idl
partial interface HTMLUListElement {
  [CEReactions] attribute boolean compact;
  [CEReactions] 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.)

```idl
partial interface Document {
  [CEReactions] attribute [LegacyNullToEmptyString] DOMString fgColor;
  [CEReactions] attribute [LegacyNullToEmptyString] DOMString linkColor;
  [CEReactions] attribute [LegacyNullToEmptyString] DOMString vlinkColor;
  [CEReactions] attribute [LegacyNullToEmptyString] DOMString alinkColor;
  [CEReactions] attribute [LegacyNullToEmptyString] DOMString bgColor;

  [SameObject] readonly attribute HTMLCollection anchors;
  [SameObject] readonly attribute HTMLCollection applets;

  undefined clear();
  undefined captureEvents();
  undefined 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.
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 nothing. (It exists for historical reasons.)

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.

```idl
IDL attribute

fgColor: text;
linkColor: link;
vlinkColor: vlink;
allinkColor: alink;
bgColor: bgcolor;
```

The partial interface `Window` defines:

```idl
partial interface Window {
  undefined captureEvents();
  undefined 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:

```idl
[Exposed=Window]
interface External {
  undefined AddSearchProvider();
  undefined IsSearchProviderInstalled();
};
```

The AddSearchProvider() and IsSearchProviderInstalled() methods must do nothing.

```idl
interface mixin NavigatorPlugins {
  [SameObject] readonly attribute PluginArray plugins;
  [SameObject] readonly attribute MimeArray mimeTypes;
  boolean javaEnabled();
};
```

`[Exposed=Window]` interface `PluginArray` defines:

```idl
interface PluginArray {
  undefined refresh();
  readonly attribute unsigned long length;
  getter object? item(unsigned long index);
  object? namedItem(DOMString name);
};
```

`[Exposed=Window]` interface `MimeTypeArray` defines:

```idl
interface MimeArray {
  readonly attribute unsigned long length;
  getter object? item(unsigned long index);
  object? namedItem(DOMString name);
};
```
We might be able to remove the Plugin\textsuperscript{1235} and MimeType\textsuperscript{1235} classes entirely: see Issue \#6317.

Each NavigatorPlugins\textsuperscript{1234} object has a plugins array, which is a new PluginArray\textsuperscript{1236} object, and a mime types array, which is a new MimeTypeArray\textsuperscript{1234} object.

The NavigatorPlugins\textsuperscript{1234} mixin's plugins getter steps are to return this's plugins array\textsuperscript{1235}.

The NavigatorPlugins\textsuperscript{1234} mixin's mimeTypes getter steps are to return this's mime types array\textsuperscript{1235}.

The NavigatorPlugins\textsuperscript{1234} mixin's javaEnabled() method steps are to return false.

The PluginArray\textsuperscript{1234} interface's length getter steps are to return 0.

The PluginArray\textsuperscript{1234} interface's item() and namedItem() method steps are to return null.

The PluginArray\textsuperscript{1234} interface's refresh() method steps are to do nothing.

The MimeTypeArray\textsuperscript{1234} interface's length getter steps are to return 0.

The MimeTypeArray\textsuperscript{1234} interface's item() and namedItem() method steps are to return null.

The Plugin\textsuperscript{1235} interface's name, description, filename, and length getter steps are to return undefined.

The Plugin\textsuperscript{1235} interface's item() and namedItem() method steps are to do nothing.

The MimeType\textsuperscript{1235} interface's type, description, suffixes, and enabledPlugin getter steps are to return undefined.
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 an ASCII case-insensitive match for the string "utf-8". [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 a 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 MIME Sniffing. [MIMESNIFF]

**File extension(s):**

"html" and "htm" are commonly, but certainly not exclusively, used as the extension for HTML documents.
### 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)  

#### 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`.  

#### 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

Fragments used with multipart/x-mixed-replace resources apply to each body part as defined by the type used by that body part.

17.3 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 document element from the HTML namespace. Thus, the relevant specifications are XML, Namespaces in XML, and this specification. [XML] [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 document 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

Fragments used with application/xhtml+xml resources have the same semantics as with any XML MIME type. [RFC7303]

17.4 text/ping

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:
ping

Required parameters:
No parameters

Optional parameters:
charset

The charset parameter may be provided. The parameter's value must be "utf-8". This parameter serves no purpose; it is only allowed for compatibility with legacy servers.

Encoding considerations:
Not applicable.

Security considerations:
If used exclusively in the fashion described in the context of hyperlink auditing, this type introduces no new security concerns.

Interoperability considerations:
Rules applicable to this type are defined in this specification.

Published specification:
This document is the relevant specification.

Applications that use this media type:
Web browsers.

Additional information:
Magic number(s):
text/ping resources always consist of the four bytes 0x50 0x49 0x4E 0x47 (`PING').

File extension(s):
No specific file extension is 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:
Only intended for use with HTTP POST requests generated as part of a web browser's processing of the ping attribute.

Author:
Ian Hickson <ian@hixie.ch>
Fragments have no meaning with text/ping resources.

17.5 application/microdata+json

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:
microdata+json

Required parameters:
Same as for application/json

Optional parameters:
Same as for application/json

Encoding considerations:
8bit (always UTF-8)

Security considerations:
Same as for application/json

Interoperability considerations:
Same as for application/json

Published specification:
Labeling a resource with the application/microdata+json type asserts that the resource is a JSON text that consists of an object with a single entry called "items" consisting of an array of entries, each of which consists of an object with an entry called "id" whose value is a string, an entry called "type" whose value is another string, and an entry called "properties" whose value is an object whose entries each have a value consisting of an array of either objects or strings, the objects being of the same form as the objects in the aforementioned "items" entry. Thus, the relevant specifications are JSON and this specification.

Applications that use this media type:
Applications that transfer data intended for use with HTML's microdata feature, especially in the context of drag-and-drop, are the primary application class for this type.

Additional information:

Magic number(s):
Same as for application/json

File extension(s):
Same as for application/json

Macintosh file type code(s):
Same as for application/json

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
17.6 text/event-stream

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:**
- event-stream

**Required parameters:**
- No parameters

**Optional parameters:**
- charset
  
  The `charset` parameter may be provided. The parameter's value must be "utf-8"). This parameter serves no purpose; it is only allowed for compatibility with legacy servers.

**Encoding considerations:**
- 8bit (always UTF-8)

**Security considerations:**
- An event stream from an origin distinct from the origin of the content consuming the event stream can result in information leakage. To avoid this, user agents are required to apply CORS semantics. [FETCH]

  Event streams can overwhelm a user agent; a user agent is expected to apply suitable restrictions to avoid depleting local resources because of an overabundance of information from an event stream.

  Servers can be overwhelmed if a situation develops in which the server is causing clients to reconnect rapidly. Servers should use a 5xx status code to indicate capacity problems, as this will prevent conforming clients from reconnecting automatically.

**Interoperability considerations:**
- Rules for processing both conforming and non-conforming content are defined in this specification.

**Published specification:**
- This document is the relevant specification.

**Applications that use this media type:**
- Web browsers and tools using web services.

**Additional information:**
- Magic number(s):
  - No sequence of bytes can uniquely identify an event stream.

- 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:**
- This format is only expected to be used by dynamic open-ended streams served using HTTP or a similar protocol. Finite resources are not expected to be labeled with this type.

**Author:**
- Ian Hickson <ian@hixie.ch>
Fragments have no meaning with text/event-stream\textsuperscript{p1241} resources.

17.7 `Cross-Origin-Embedder-Policy`\textsuperscript{§p1242}

This section describes a header for registration in the Permanent Message Header Field Registry. [RFC3864]\textsuperscript{p1275}

**Header field name:**
Cross-Origin-Embedder-Policy

**Applicable protocol:**
http

**Status:**
standard

**Author/Change controller:**
WHATWG

**Specification document(s):**
This document is the relevant specification.

**Related information:**
None.

17.8 `Cross-Origin-Embedder-Policy-Report-Only`\textsuperscript{§p1242}

This section describes a header for registration in the Permanent Message Header Field Registry. [RFC3864]\textsuperscript{p1275}

**Header field name:**
Cross-Origin-Embedder-Policy-Report-Only

**Applicable protocol:**
http

**Status:**
standard

**Author/Change controller:**
WHATWG

**Specification document(s):**
This document is the relevant specification.

**Related information:**
None.

17.9 `Cross-Origin-Opener-Policy`\textsuperscript{§p1242}

This section describes a header for registration in the Permanent Message Header Field Registry. [RFC3864]\textsuperscript{p1275}

**Header field name:**
Cross-Origin-Opener-Policy

**Applicable protocol:**
http
This section describes a header for registration in the Permanent Message Header Field Registry. [RFC3864][43]

**17.10 `Cross-Origin-Opener-Policy-Report-Only`**

This section describes a header for registration in the Permanent Message Header Field Registry. [RFC3864][43]

**Header field name:**
Cross-Origin-Opener-Policy-Report-Only

**Applicable protocol:**
http

**Status:**
standard

**Author/Change controller:**
WHATWG

**Specification document(s):**
This document is the relevant specification.

**Related information:**
None.

---

This section describes a header for registration in the Permanent Message Header Field Registry. [RFC3864][43]

**17.11 `Origin-Agent-Cluster`**

This section describes a header for registration in the Permanent Message Header Field Registry. [RFC3864][43]

**Header field name:**
Origin-Agent-Cluster

**Applicable protocol:**
http

**Status:**
standard

**Author/Change controller:**
WHATWG

**Specification document(s):**
This document is the relevant specification.

**Related information:**
None.
17.12 `Ping-From` §p12

This section describes a header for registration in the Permanent Message Header Field Registry. [RFC3864]§1275

Header field name:

Ping-From

Applicable protocol:

http

Status:

standard

Author/Change controller:

W3C

Specification document(s):

This document is the relevant specification.

Related information:

None.

17.13 `Ping-To` §p12

This section describes a header for registration in the Permanent Message Header Field Registry. [RFC3864]§1275

Header field name:

Ping-To

Applicable protocol:

http

Status:

standard

Author/Change controller:

W3C

Specification document(s):

This document is the relevant specification.

Related information:

None.

17.14 `Refresh` §p12

This section describes a header for registration in the Permanent Message Header Field Registry. [RFC3864]§1275

Header field name:

Refresh

Applicable protocol:

http

Status:

standard

Author/Change controller:

WHATWG

Specification document(s):

This document is the relevant specification.
Related information:
None.

17.15 `Last-Event-ID`
This section describes a header for registration in the Permanent Message Header Field Registry. [RFC3864]

Header field name:
Last-Event-ID

Applicable protocol:
http

Status:
standard

Author/Change controller:
W3C

Specification document(s):
This document is the relevant specification.

Related information:
None.

17.16 `X-Frame-Options`
This section describes a header for registration in the Permanent Message Header Field Registry. [RFC3864]

Header field name:
X-Frame-Options

Applicable protocol:
http

Status:
standard

Author/Change controller:
WHATWG

Specification document(s):
This document is the relevant specification.

Related information:
None.

17.17 `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., HTTP). 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 handlers, HTML Living Standard: https://html.spec.whatwg.org/#custom-handlers
This section is non-normative.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Categories</th>
<th>Parents</th>
<th>Children</th>
<th>Attributes</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Hyperlink</td>
<td>flow, phrasing, interactive, palpable</td>
<td>phrasing</td>
<td>transparent</td>
<td>globals, href, target, download, ping, rel, hreflang, type, referrerpolicy</td>
<td>HTMLAnchorElement</td>
</tr>
<tr>
<td>abbr</td>
<td>Abbreviation</td>
<td>flow, phrasing, palpable</td>
<td>phrasing</td>
<td>phrasing</td>
<td>globals</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>address</td>
<td>Contact information for a page or article element</td>
<td>flow, palpable</td>
<td>flow</td>
<td>flow</td>
<td>globals</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>area</td>
<td>Hyperlink or dead area on an image map</td>
<td>flow, phrasing</td>
<td>phrasing</td>
<td>empty</td>
<td>globals, alt, coords, shape, href, target, download, ping, rel, referrerpolicy</td>
<td>HTMLAreaElement</td>
</tr>
<tr>
<td>article</td>
<td>Self-contained syndicatable or reusable composition</td>
<td>flow, sectioning, palpable</td>
<td>flow</td>
<td>flow</td>
<td>globals</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>aside</td>
<td>Sidebar for tangentially related content</td>
<td>flow, sectioning, palpable</td>
<td>flow</td>
<td>flow</td>
<td>globals</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>audio</td>
<td>Audio player</td>
<td>flow, phrasing, embedded, interactive, palpable</td>
<td>phrasing</td>
<td>source</td>
<td>globals, src, crossorigin, track, preload, autoplay, loop, muted, controls</td>
<td>HTMLAudioElement</td>
</tr>
<tr>
<td>b</td>
<td>Keywords</td>
<td>flow, phrasing, palpable</td>
<td>phrasing</td>
<td>phrasing</td>
<td>globals</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>base</td>
<td>Base URL and default target browsing context for hyperlinks and forms</td>
<td>metadata</td>
<td>head</td>
<td>empty</td>
<td>globals, href, target</td>
<td>HTMLBaseElement</td>
</tr>
<tr>
<td>bdi</td>
<td>Text directionality isolation</td>
<td>flow, phrasing, palpable</td>
<td>phrasing</td>
<td>phrasing</td>
<td>globals</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>bdo</td>
<td>Text directionality formatting</td>
<td>flow, phrasing, palpable</td>
<td>phrasing</td>
<td>phrasing</td>
<td>globals</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>blockquote</td>
<td>A section quoted from another source</td>
<td>flow, sectioning, root, palpable</td>
<td>flow</td>
<td>flow</td>
<td>globals, cite</td>
<td>HTMLQuoteElement</td>
</tr>
<tr>
<td>body</td>
<td>Document</td>
<td>sectioning</td>
<td>HTML</td>
<td>flow</td>
<td>globals, onafterprint</td>
<td>HTMLBodyElement</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
<td>Categories</td>
<td>Parents†</td>
<td>Children</td>
<td>Attributes</td>
<td>Interface</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------------------------</td>
<td>-----------------------------------------------</td>
<td>----------</td>
<td>----------------</td>
<td>----------------------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>body</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>onbeforeprint, onbeforeunload, onhashchange, onlanguagechange, onmessage, onmessageerror, offline, ononline, onbeforeprint, onbeforeunload, onhashchange, onlanguagechange, onmessage, onmessageerror, offline, ononline, onpageshow, onpopstate, unrejectionhandled, unstorage, onunhandledrejection, onunload</td>
<td>HTMLBodyElement</td>
</tr>
<tr>
<td><em>br</em></td>
<td>Line break, e.g. in poem or postal address</td>
<td></td>
<td></td>
<td>phrasing, empty</td>
<td>globals</td>
<td>HTMLBlockElement</td>
</tr>
<tr>
<td>button</td>
<td>Button control</td>
<td></td>
<td></td>
<td>phrasing, phrasing</td>
<td>globals: disabled, form, formaction, formmethod, formenctype, formnovalidate, formtarget, labelable, listed, named, palpable, associated, form, submittable, labelable, associated, form, palpable</td>
<td>HTMLButtonElement</td>
</tr>
<tr>
<td>canvas</td>
<td>Scriptable bitmap canvas</td>
<td></td>
<td></td>
<td>phrasing, transparent</td>
<td>globals: width, height</td>
<td>HTMLCanvasElement</td>
</tr>
<tr>
<td>caption</td>
<td>Table caption</td>
<td></td>
<td></td>
<td>flow, globals</td>
<td>globals</td>
<td>HTMLTableCaptionElement</td>
</tr>
<tr>
<td>cite</td>
<td>Title of a work</td>
<td></td>
<td></td>
<td>phrasing, globals</td>
<td>globals</td>
<td>HTMLBlockElement</td>
</tr>
<tr>
<td>code</td>
<td>Computer code</td>
<td></td>
<td></td>
<td>phrasing, globals</td>
<td>globals</td>
<td>HTMLBlockElement</td>
</tr>
<tr>
<td>col</td>
<td>Table column</td>
<td></td>
<td></td>
<td>colgroup, globals</td>
<td>span</td>
<td>HTMLTableColElement</td>
</tr>
<tr>
<td>colgroup</td>
<td>Group of columns in a table</td>
<td></td>
<td></td>
<td>table, globals</td>
<td>span</td>
<td>HTMLTableColElement</td>
</tr>
<tr>
<td>data</td>
<td>Machine-readable equivalent</td>
<td></td>
<td></td>
<td>phrasing, globals</td>
<td>value</td>
<td>HTMLDataElement</td>
</tr>
<tr>
<td>datalist</td>
<td>Container for options for combo box control</td>
<td></td>
<td></td>
<td>phrasing, globals</td>
<td>option, script, supporting elements</td>
<td>HTMLDataListElement</td>
</tr>
<tr>
<td>dd</td>
<td>Content for corresponding dt element(s)</td>
<td></td>
<td></td>
<td>gl, div, flow</td>
<td>globals</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>det</td>
<td>A removal from the document</td>
<td></td>
<td></td>
<td>flow, transparent</td>
<td>globals, cite, date, datetime</td>
<td>HTMLModElement</td>
</tr>
<tr>
<td>details</td>
<td>Disclosure control for hiding details</td>
<td></td>
<td></td>
<td>flow, summary</td>
<td>globals, open</td>
<td>HTMLDetailsElement</td>
</tr>
<tr>
<td>dfn</td>
<td>Defining instance</td>
<td></td>
<td></td>
<td>phrasing, globals</td>
<td>globals</td>
<td>HTMLBlockElement</td>
</tr>
<tr>
<td>dialog</td>
<td>Dialog box or window</td>
<td></td>
<td></td>
<td>flow, globals</td>
<td>open</td>
<td>HTMLDialogElement</td>
</tr>
<tr>
<td>div</td>
<td>Generic flow container, or container for name-value groups in gl</td>
<td></td>
<td></td>
<td>flow, gl, flow</td>
<td>globals</td>
<td>HTMLDivElement</td>
</tr>
</tbody>
</table>

HTMLDivElement, HTMLDialogElement, HTMLElement, HTMLDetailsElement, HTMLDataElement, HTMLDataListElement, HTMLCanvasElement, HTMLButtonElement, HTMLTableCaptionElement, HTMLTableColElement, HTMLBlockElement, HTMLTableElement, HTMLModElement, HTMLDetailsElement, HTMLDataListElement, HTMLCanvasElement, HTMLTableCaptionElement, HTMLTableColElement, HTMLBlockElement, HTMLTableElement, HTMLModElement.
<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Categories</th>
<th>Parents†</th>
<th>Children</th>
<th>Attributes</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dl</code></td>
<td>Association list consisting of zero or more name-value groups</td>
<td><code>flow</code>, <code>palpable</code></td>
<td><code>flow</code></td>
<td><code>dt</code>, <code>di</code></td>
<td><code>globals</code></td>
<td><code>HTMLListElement</code></td>
</tr>
<tr>
<td><code>dt</code></td>
<td>Legend for corresponding <code>dd</code> element(s)</td>
<td>none</td>
<td><code>dl</code>, <code>di</code></td>
<td><code>flow</code></td>
<td><code>globals</code></td>
<td><code>HTMLElement</code></td>
</tr>
<tr>
<td><code>em</code></td>
<td>Stress emphasis</td>
<td><code>flow</code>, <code>phrasing</code>, <code>palpable</code></td>
<td><code>flow</code></td>
<td><code>phrasing</code></td>
<td><code>globals</code></td>
<td><code>HTMLElement</code></td>
</tr>
<tr>
<td><code>embed</code></td>
<td>Plugin</td>
<td><code>flow</code>, <code>phrasing</code>, <code>embedded</code>, <code>interactive</code>, <code>palpable</code></td>
<td><code>flow</code></td>
<td><code>phrasing</code></td>
<td><code>globals</code></td>
<td><code>HTMLEmbedElement</code></td>
</tr>
<tr>
<td><code>fieldset</code></td>
<td>Group of form controls</td>
<td><code>flow</code>, <code>sectioning root</code>, <code>listed</code>, <code>form-associated</code>, <code>palpable</code></td>
<td><code>flow</code></td>
<td><code>legend</code></td>
<td><code>globals</code>, <code>disabled</code>, <code>form</code>, <code>name</code></td>
<td><code>HTMLFieldSetElement</code></td>
</tr>
<tr>
<td><code>figcaption</code></td>
<td>Caption for <code>figure</code></td>
<td>none</td>
<td><code>figure</code></td>
<td><code>flow</code></td>
<td><code>globals</code></td>
<td><code>HTMLElement</code></td>
</tr>
<tr>
<td><code>figure</code></td>
<td>Figure with optional caption</td>
<td><code>flow</code>, <code>sectioning root</code>, <code>palpable</code></td>
<td><code>flow</code></td>
<td><code>figcaption</code></td>
<td><code>globals</code></td>
<td><code>HTMLElement</code></td>
</tr>
<tr>
<td><code>footer</code></td>
<td>Footer for a page or section</td>
<td><code>flow</code>, <code>palpable</code></td>
<td><code>flow</code></td>
<td><code>flow</code></td>
<td><code>globals</code></td>
<td><code>HTMLElement</code></td>
</tr>
<tr>
<td><code>form</code></td>
<td>User-submittable form</td>
<td><code>flow</code>, <code>palpable</code></td>
<td><code>flow</code></td>
<td><code>flow</code></td>
<td><code>globals</code>, <code>accept-charset</code>, <code>action</code>, <code>autocomplete</code>, <code>encType</code>, <code>method</code>, <code>name</code>, <code>novalidate</code>, <code>referrerPolicy</code>, <code>target</code></td>
<td><code>HTMLFormElement</code></td>
</tr>
<tr>
<td><code>h1</code>, <code>h2</code>, <code>h3</code>, <code>h4</code>, <code>h5</code>, <code>h6</code></td>
<td>Section heading</td>
<td><code>flow</code>, <code>heading</code>, <code>summary</code>, <code>palpable</code></td>
<td><code>flow</code>, <code>legend</code>, <code>summary</code>, <code>flow</code></td>
<td><code>phrasing</code></td>
<td><code>globals</code></td>
<td><code>HTMLHeadingElement</code></td>
</tr>
<tr>
<td><code>head</code></td>
<td>Container for document metadata</td>
<td>none</td>
<td><code>html</code></td>
<td><code>metadata content</code></td>
<td><code>globals</code></td>
<td><code>HTMLHeadElement</code></td>
</tr>
<tr>
<td><code>header</code></td>
<td>Introductory or navigational aids for a page or section</td>
<td><code>flow</code>, <code>palpable</code></td>
<td><code>flow</code></td>
<td><code>flow</code></td>
<td><code>globals</code></td>
<td><code>HTMLElement</code></td>
</tr>
<tr>
<td><code>hgroup</code></td>
<td>Heading group</td>
<td><code>flow</code>, <code>heading</code>, <code>palpable</code></td>
<td><code>flow</code>, <code>legend</code>, <code>summary</code>, <code>flow</code></td>
<td><code>h1</code>, <code>h2</code>, <code>h3</code>, <code>h4</code>, <code>h5</code>, <code>h6</code>, <code>script</code>, <code>supporting elements</code></td>
<td><code>globals</code></td>
<td><code>HTMLElement</code></td>
</tr>
<tr>
<td><code>hr</code></td>
<td>Thematic break</td>
<td><code>flow</code></td>
<td><code>flow</code></td>
<td>empty</td>
<td><code>globals</code></td>
<td><code>HTMLHorizontalRuleElement</code></td>
</tr>
<tr>
<td><code>html</code></td>
<td>Root element</td>
<td>none</td>
<td>none</td>
<td><code>head</code>, <code>body</code></td>
<td><code>globals</code>, <code>manifest</code></td>
<td><code>HTMLHtmlElement</code></td>
</tr>
<tr>
<td><code>i</code></td>
<td>Alternate voice</td>
<td><code>flow</code>, <code>phrasing</code>, <code>palpable</code></td>
<td><code>flow</code>, <code>phrasing</code></td>
<td><code>phrasing</code></td>
<td><code>globals</code></td>
<td><code>HTMLElement</code></td>
</tr>
<tr>
<td><code>iframe</code></td>
<td>Nested browsing context</td>
<td><code>flow</code>, <code>phrasing</code>, <code>embedded</code>, <code>interactive</code>, <code>palpable</code></td>
<td><code>flow</code>, <code>phrasing</code></td>
<td>empty</td>
<td><code>globals</code>, <code>src</code>, <code>srcdoc</code>, <code>name</code>, <code>sandbox</code>, <code>allow</code>, <code>allowfullscreen</code>, <code>width</code>, <code>height</code>, <code>referrerPolicy</code>, <code>loading</code></td>
<td><code>HTMLIFrameElement</code></td>
</tr>
<tr>
<td><code>img</code></td>
<td>Image</td>
<td><code>flow</code>, <code>phrasing</code></td>
<td><code>flow</code>, <code>phrasing</code></td>
<td>empty</td>
<td><code>globals</code>, <code>alt</code>, <code>src</code>, <code>srcset</code></td>
<td><code>HTMLImageElement</code></td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
<td>Categories</td>
<td>Parents‡</td>
<td>Children‡</td>
<td>Attributes†</td>
<td>Interface‡</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>------------</td>
<td>----------</td>
<td>-----------</td>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>input</td>
<td>Form control</td>
<td>flow; phrasing; interactive; form-associated; palpable; globals</td>
<td>phrasing; empty</td>
<td>globals: accept; alt; autocomplete; checked; dirname; disabled; form; for; formaction; formtype; formmethod; formnovalidate; formtarget; height; list; max; max-length; min; min-length; multiple; name; pattern; placeholder; required; size; src; readonly; step; type; value; width; varies*; points; ranges; sliding; textbox; toggle; type; vertical-align; vertical-offset; vertical; horizontal-align; horizontal-offset; horizontal; wrap; wrap-after; wrap-before; wrap-inside; wrap-outside; wrap-right</td>
<td>HTMLInputElement; HTMLSelectElement; HTMLTextAreaElement</td>
<td></td>
</tr>
<tr>
<td>img</td>
<td>An addition to the document</td>
<td>flow; phrasing; interactive; form-associated; palpable</td>
<td>phrasing; transparent</td>
<td>globals: cite; datetime</td>
<td>HTMLModElement</td>
<td></td>
</tr>
<tr>
<td>kbd</td>
<td>User input</td>
<td>flow; phrasing; interactive; form-associated; palpable</td>
<td>phrasing; *</td>
<td>globals:</td>
<td>HTMLInputElement; HTMLSelectElement; HTMLTextAreaElement</td>
<td></td>
</tr>
<tr>
<td>label</td>
<td>Caption for a form control</td>
<td>flow; phrasing; interactive; form-associated; palpable</td>
<td>phrasing; *</td>
<td>globals: for</td>
<td>HTMLLabelElement</td>
<td></td>
</tr>
<tr>
<td>legend</td>
<td>Caption for fieldset</td>
<td>none; fieldset</td>
<td>phrasing; heading content</td>
<td>globals:</td>
<td>HTMLLegendElement</td>
<td></td>
</tr>
<tr>
<td>li</td>
<td>List item</td>
<td>none</td>
<td>phrasing; flow</td>
<td>globals: value</td>
<td>HTMLItemElement; HTMLOListElement</td>
<td></td>
</tr>
<tr>
<td>link</td>
<td>Link metadata</td>
<td>metadata; head; noscript</td>
<td>empty</td>
<td>globals: href; rel; rel='stylesheet'; media; hreflang; type; sizes; imagesizes; alt; cite; charset; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decoding; decod...</td>
<td>HTMLElement; HTMLSelectElement; HTMLTextAreaElement</td>
<td></td>
</tr>
<tr>
<td>main</td>
<td>Container for the dominant contents of the document</td>
<td>flow; palpable; *</td>
<td>flow; *</td>
<td>globals:</td>
<td>HTMLElement</td>
<td></td>
</tr>
<tr>
<td>map</td>
<td>Image</td>
<td>flow; phrasing; embededdef; interactive; form-associated; palpable</td>
<td>phrasing; transparent; area</td>
<td>globals: name;</td>
<td>HTMLMapElement; HTMLOListElement</td>
<td></td>
</tr>
<tr>
<td>mark</td>
<td>Highlight</td>
<td>flow; phrasing; interactive; palpable</td>
<td>phrasing; *</td>
<td>globals:</td>
<td>HTMLElement</td>
<td></td>
</tr>
<tr>
<td>MathML</td>
<td>MathML root</td>
<td>flow; phrasing; embededdef; interactive; form-associated; palpable; globals</td>
<td>phrasing; per [MathML; per [MathML]</td>
<td>globals:</td>
<td>Element</td>
<td></td>
</tr>
<tr>
<td>menu</td>
<td>Menu of commands</td>
<td>flow; palpable; *</td>
<td>flow; li; script supporting elements</td>
<td>globals:</td>
<td>HTMLMenuElement</td>
<td></td>
</tr>
<tr>
<td>meta</td>
<td>Text metadata</td>
<td>metadata; flow; head; noscript; phrasing</td>
<td>empty</td>
<td>globals: name; http-equiv; content; charset;</td>
<td>HTMLMetaElement</td>
<td></td>
</tr>
<tr>
<td>meter</td>
<td>Gauge</td>
<td>flow; phrasing; labelable; palpable; *</td>
<td>phrasing; *</td>
<td>globals: value; min; max; low; high; optimum;</td>
<td>HTMLMeterElement</td>
<td></td>
</tr>
<tr>
<td>nav</td>
<td>Section with navigational links</td>
<td>flow; sectioning; palpable; *</td>
<td>flow; *</td>
<td>globals:</td>
<td>HTMLElement</td>
<td></td>
</tr>
<tr>
<td>noscript</td>
<td>Fallback content for script</td>
<td>metadata; flow; phrasing</td>
<td>head; varies*</td>
<td>globals:</td>
<td>HTMLElement</td>
<td></td>
</tr>
<tr>
<td>object</td>
<td>Image,</td>
<td>flow; phrasing</td>
<td>param; *;</td>
<td>globals: data; type; name;</td>
<td>HTMLObjectElement; HTMLOListElement; HTMLSelectElement; HTMLTextAreaElement; HTMLElement; HTMLMapElement; HTMLOListElement; HTMLSelectElement; HTMLTextAreaElement; HTMLElement; HTMLMenuElement; HTMLMetaElement; HTMLMeterElement; HTMLElement; HTMLElement</td>
<td></td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
<td>Categories</td>
<td>Parents†</td>
<td>Children</td>
<td>Attributes</td>
<td>Interface</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>------------</td>
<td>----------</td>
<td>----------</td>
<td>------------</td>
<td>-----------</td>
</tr>
<tr>
<td>ol</td>
<td>Ordered list</td>
<td>flow, palpable</td>
<td>flow</td>
<td>li, script-supporting elements</td>
<td>globals; reversed, start, type</td>
<td>HTMLListElement</td>
</tr>
<tr>
<td>optgroup</td>
<td>Group of options in a list box</td>
<td>none</td>
<td>select</td>
<td>option, script-supporting elements</td>
<td>globals; disabled, label</td>
<td>HTMLSelectElement</td>
</tr>
<tr>
<td>option</td>
<td>Option in a list box or combo box control</td>
<td>none</td>
<td>select, datalist, optgroup</td>
<td>text</td>
<td>globals; disabled, label, selected, value</td>
<td>HTMLSelectElement</td>
</tr>
<tr>
<td>output</td>
<td>Calculated output value</td>
<td>flow, phrasing, listed, labelable, form-associated, palpable</td>
<td>phrasing, palpable</td>
<td>globals; form, name</td>
<td>HTMLOutputElement</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>Paragraph</td>
<td>flow, palpable</td>
<td>flow</td>
<td>phrasing</td>
<td>globals</td>
<td>HTMLParagraphElement</td>
</tr>
<tr>
<td>param</td>
<td>Parameter for object</td>
<td>none</td>
<td>object</td>
<td>empty</td>
<td>globals; name, value</td>
<td>HTMLParamElement</td>
</tr>
<tr>
<td>picture</td>
<td>Image</td>
<td>flow, phrasing, embed</td>
<td>source, one</td>
<td>globals</td>
<td>HTMLPictureElement</td>
<td></td>
</tr>
<tr>
<td>pre</td>
<td>Block of preformatted text</td>
<td>flow, palpable</td>
<td>flow</td>
<td>phrasing</td>
<td>globals</td>
<td>HTMLPreElement</td>
</tr>
<tr>
<td>progress</td>
<td>Progress bar</td>
<td>flow, phrasing, embed</td>
<td>phrasing, palpable</td>
<td>globals; value, max</td>
<td>HTMLProgressElement</td>
<td></td>
</tr>
<tr>
<td>q</td>
<td>Quotation</td>
<td>flow, palpable</td>
<td>flow</td>
<td>phrasing</td>
<td>globals; cite</td>
<td>HTMLQuoteElement</td>
</tr>
<tr>
<td>rp</td>
<td>Parenthesis for ruby annotation text</td>
<td>none</td>
<td>ruby</td>
<td>text, globals</td>
<td>HTMLParagraphElement</td>
<td></td>
</tr>
<tr>
<td>rt</td>
<td>Ruby annotation text</td>
<td>none</td>
<td>ruby</td>
<td>phrasing, globals</td>
<td>HTMLParagraphElement</td>
<td></td>
</tr>
<tr>
<td>ruby</td>
<td>Ruby annotation(s)</td>
<td>flow, phrasing, palpable</td>
<td>phrasing, rt, rp</td>
<td>globals</td>
<td>HTMLParagraphElement</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>Inaccurate text</td>
<td>flow, phrasing, palpable</td>
<td>phrasing</td>
<td>globals</td>
<td>HTMLParagraphElement</td>
<td></td>
</tr>
<tr>
<td>samp</td>
<td>Computer output</td>
<td>flow, phrasing, palpable</td>
<td>phrasing</td>
<td>globals</td>
<td>HTMLParagraphElement</td>
<td></td>
</tr>
<tr>
<td>script</td>
<td>Embedded script</td>
<td>metadata, head, script, data, or script documentation</td>
<td>script, supporting</td>
<td>script; src, type, async, defer, crossOrigin, integrity, referrerPolicy</td>
<td>HTMLScriptElement</td>
<td></td>
</tr>
<tr>
<td>section</td>
<td>Generic document or application</td>
<td>flow, sectioning, palpable</td>
<td>flow</td>
<td>globals</td>
<td>HTMLParagraphElement</td>
<td></td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
<td>Categories</td>
<td>Parents†</td>
<td>Children</td>
<td>Attributes</td>
<td>Interface</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------</td>
<td>-------------------------------------------------</td>
<td>----------</td>
<td>-----------------------------------------------</td>
<td>-------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>select</td>
<td>1️⃣ List box control</td>
<td>flow, phrasing, interactive, listed, labelable, submittable, selectable, palpable</td>
<td>phrasing</td>
<td>option, optgroup, script, supporting elements</td>
<td>globals, autocomplete, disabled, form, multiple, name, required, size</td>
<td>HTMLSelectElement</td>
</tr>
<tr>
<td>slot</td>
<td>2️⃣ Shadow tree slot</td>
<td>flow, phrasing, palpable</td>
<td>phrasing</td>
<td>transparent</td>
<td>globals, name</td>
<td>HTMLSlotElement</td>
</tr>
<tr>
<td>small</td>
<td>3️⃣ Side comment</td>
<td>flow, phrasing, palpable</td>
<td>phrasing</td>
<td>phrasing</td>
<td>globals</td>
<td>HTMLSpanElement</td>
</tr>
<tr>
<td>source</td>
<td>4️⃣ Image source for img or media source for video or audio</td>
<td>none</td>
<td>picture, video, audio</td>
<td>empty</td>
<td>globals, src, type, srcset, sizes, media</td>
<td>HTMLSourceElement</td>
</tr>
<tr>
<td>span</td>
<td>5️⃣ Generic phrasing container</td>
<td>flow, phrasing, palpable</td>
<td>phrasing</td>
<td>phrasing</td>
<td>globals</td>
<td>HTMLSpanElement</td>
</tr>
<tr>
<td>strong</td>
<td>6️⃣ Importance</td>
<td>flow, phrasing, palpable</td>
<td>phrasing</td>
<td>phrasing</td>
<td>globals</td>
<td>HTMLSpanElement</td>
</tr>
<tr>
<td>style</td>
<td>7️⃣ Embedded styling information</td>
<td>metadata</td>
<td>head, noscript, text*</td>
<td>text*</td>
<td>globals, media, text*</td>
<td>HTMLStyleElement</td>
</tr>
<tr>
<td>sub</td>
<td>8️⃣ Subscript</td>
<td>flow, phrasing, palpable</td>
<td>phrasing</td>
<td>phrasing</td>
<td>globals</td>
<td>HTMLSpanElement</td>
</tr>
<tr>
<td>summary</td>
<td>9️⃣ Caption for details</td>
<td>none</td>
<td>details</td>
<td>phrasing, heading content</td>
<td>globals</td>
<td>HTMLSpanElement</td>
</tr>
<tr>
<td>sup</td>
<td>1️⃣ Superscript</td>
<td>flow, phrasing, palpable</td>
<td>phrasing</td>
<td>phrasing</td>
<td>globals</td>
<td>HTMLSpanElement</td>
</tr>
<tr>
<td>SVG svg</td>
<td>1️⃣ SVG root</td>
<td>flow, phrasing, embedded, palpable</td>
<td>phrasing</td>
<td>per [SVG]</td>
<td>per [SVG]</td>
<td>SVGSVGElement</td>
</tr>
<tr>
<td>table</td>
<td>1️⃣ Table</td>
<td>flow, palpable</td>
<td>flow</td>
<td>caption, colgroup, thead, tbody, tfoot, tr, script, supporting elements</td>
<td>globals</td>
<td>HTMLTableElement</td>
</tr>
<tr>
<td>tbody</td>
<td>1️⃣ Group of rows in a table</td>
<td>none</td>
<td>table</td>
<td>tr, script, supporting elements</td>
<td>globals</td>
<td>HTMLTableSectionElement</td>
</tr>
<tr>
<td>td</td>
<td>2️⃣ Table cell</td>
<td>sectioning root</td>
<td>tr</td>
<td>flow</td>
<td>globals, colspan, rowspan</td>
<td>HTMLTableCellElement</td>
</tr>
<tr>
<td>template</td>
<td>3️⃣ Template</td>
<td>metadata, phrasing, script, supporting</td>
<td>metadata</td>
<td>empty</td>
<td>globals</td>
<td>HTMLTemplateElement</td>
</tr>
<tr>
<td>textarea</td>
<td>4️⃣ Multiline text controls</td>
<td>flow, phrasing, interactive, listed, labelable, submittable,</td>
<td>phrasing</td>
<td>text</td>
<td>globals, cols, dirname, disabled, form, maxlength, minlength, name, placeholder, readonly, required, rows, wrap</td>
<td>HTMTextAreElement</td>
</tr>
</tbody>
</table>
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 \( \texttt{a} \) element’s “Parents” column says “phrasing”, so any element whose content model contains the “phrasing” category could be a parent of an \( \texttt{a} \) element. Since the “flow” category includes all the “phrasing” elements, that means the \( \texttt{th} \) element could be a parent to an \( \texttt{a} \) element.

---

**Element content categories**

**This section is non-normative.**

<table>
<thead>
<tr>
<th>Category</th>
<th>Elements</th>
<th>Elements with exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metadata content (^{312})</td>
<td>base(^{312}), link(^{312}), meta(^{312}), noscript(^{312}), script(^{312}), style(^{312}), template(^{312}), title(^{312})</td>
<td>(\text{area}^{312}) (if it is a descendant of a (\texttt{map}^{312}) element); (\text{link}^{312}) (if it is allowed in the (\texttt{body}^{312})); (\text{main}^{312}) (if it is a hierarchically correct main element(^{312})); (\text{meta}^{312}) (if the (\texttt{itemprop}^{312}) attribute is present)</td>
</tr>
<tr>
<td>Flow content (^{312})</td>
<td>a(^{312}), address(^{312}), article(^{312}), aside(^{312}), button(^{312}), canvas(^{312}), cite(^{312}), code(^{312}), data(^{312}), data-list(^{312}), details(^{312}), dfn(^{312}), dialog(^{312}), div(^{312}), dl(^{312}), dt(^{312}), em(^{312}), embed(^{312}), fieldset(^{312}), figure(^{312}), footer(^{312}), form(^{312}), h1(^{312}), h2(^{312}), h3(^{312}), h4(^{312}), h5(^{312}), h6(^{312}), header(^{312}), hr(^{312})</td>
<td>(\text{area}^{312}) (if it is a descendant of a (\texttt{map}^{312}) element); (\text{link}^{312}) (if it is allowed in the (\texttt{body}^{312})); (\text{main}^{312}) (if it is a hierarchically correct main element(^{312})); (\text{meta}^{312}) (if the (\texttt{itemprop}^{312}) attribute is present)</td>
</tr>
<tr>
<td>Category</td>
<td>Elements</td>
<td>Elements with exceptions</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>This section is non-normative.</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Element(s)</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>abbr</td>
<td>th</td>
<td>Alternative label to use for</td>
<td>Text</td>
</tr>
</tbody>
</table>

This section is non-normative.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Element(s)</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>accesskey</td>
<td>HTML elements[^79]</td>
<td>Keyboard shortcut to activate or focus element</td>
<td>Ordered set of unique space-separated tokens[^79], none of which are identical to another, each consisting of one code point in length</td>
</tr>
<tr>
<td>allow</td>
<td>iframe[^64]</td>
<td>Permissions policy to be applied to the iframe[^64]’s contents</td>
<td>Serialized permissions policy</td>
</tr>
<tr>
<td>allowfullscreen</td>
<td>iframe[^64]</td>
<td>Whether to allow the iframe[^64]’s contents to use requestFullscreen()</td>
<td>Boolean attribute[^64]</td>
</tr>
<tr>
<td>alt</td>
<td>area[^81]; img[^112]; input[^114]</td>
<td>Replacement text for use when images are not available</td>
<td>Text[^114]</td>
</tr>
<tr>
<td>as</td>
<td>link[^61]</td>
<td>Potential destination for a preload request (for rel[^61]=&quot;preload[^61]&quot; and rel[^61]=&quot;modulepreload[^61]&quot;)</td>
<td>Potential destination, for rel[^61]=&quot;preload[^61]&quot;; script-like destination, for rel[^61]=&quot;modulepreload[^61]&quot;</td>
</tr>
<tr>
<td>async</td>
<td>script[^61]</td>
<td>Execute script when available, without blocking while fetching</td>
<td>Boolean attribute[^61]</td>
</tr>
<tr>
<td>autocapitalize</td>
<td>HTML elements[^79]</td>
<td>Recommended autocapitlization behavior (for supported input methods)</td>
<td>*on[^79]; *off[^79]; none[^79]; sentences[^79]; words[^79]; characters[^79]</td>
</tr>
<tr>
<td>autocomplete</td>
<td>form[^15]</td>
<td>Default setting for autofill feature for controls in the form</td>
<td>&quot;on&quot;; &quot;off&quot;</td>
</tr>
<tr>
<td>autocomplete</td>
<td>input[^15]; select[^17]; textarea[^17]</td>
<td>Hint for form autofill feature for controls in the form</td>
<td>Autocomplete field[^17] name and related tokens*</td>
</tr>
<tr>
<td>Attribute</td>
<td>Element(s)</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>decoding</td>
<td>img</td>
<td>Decoding hint to use when processing this image for presentation</td>
<td>&quot;<em>sync</em>&quot;, &quot;async&quot;, &quot;auto&quot;</td>
</tr>
<tr>
<td>default</td>
<td>track</td>
<td>Enable the track if no other text track is more suitable</td>
<td>Boolean attribute</td>
</tr>
<tr>
<td>defer</td>
<td>script</td>
<td>Defer script execution</td>
<td>Boolean attribute</td>
</tr>
<tr>
<td>dir</td>
<td>HTML elements</td>
<td>The text directionality of the element</td>
<td>&quot;ltr&quot;, &quot;rtl&quot;, &quot;auto&quot;</td>
</tr>
<tr>
<td>dir</td>
<td>bdo</td>
<td>The text directionality of the element</td>
<td>&quot;ltr&quot;, &quot;rtl&quot;, &quot;auto&quot;</td>
</tr>
<tr>
<td>dirname</td>
<td>input; textarea</td>
<td>Name of form control to use for sending the element's directionality in form submission</td>
<td>Text</td>
</tr>
<tr>
<td>disabled</td>
<td>button; input; optgroup; option; select; textarea; form-associated custom elements</td>
<td>Whether the form control is disabled</td>
<td>Boolean attribute</td>
</tr>
<tr>
<td>disabled</td>
<td>fieldset</td>
<td>Whether the descendant form controls, except any inside legend, are disabled</td>
<td>Boolean attribute</td>
</tr>
<tr>
<td>disabled</td>
<td>link</td>
<td>Whether the link is disabled</td>
<td>Boolean attribute</td>
</tr>
<tr>
<td>download</td>
<td>a; area</td>
<td>Whether to download the resource instead of navigating to it, and its file name if so</td>
<td>Text</td>
</tr>
<tr>
<td>draggable</td>
<td>HTML elements</td>
<td>Whether the element is draggable</td>
<td>&quot;true&quot;, &quot;false&quot;</td>
</tr>
<tr>
<td>enctype</td>
<td>form</td>
<td>Entry list encoding type to use for form submission</td>
<td>&quot;text/plain&quot;, &quot;multipart/form-data&quot;, &quot;application/x-www-form-urlencoded&quot;</td>
</tr>
<tr>
<td>enterkeyhint</td>
<td>HTML elements</td>
<td>Hint for selecting an enter key action</td>
<td>&quot;enter&quot;, &quot;done&quot;, &quot;go&quot;, &quot;next&quot;, &quot;previous&quot;, &quot;search&quot;, &quot;send&quot;</td>
</tr>
<tr>
<td>for</td>
<td>label</td>
<td>Associate the label with form control</td>
<td>ID*</td>
</tr>
<tr>
<td>for</td>
<td>output</td>
<td>Specifies controls from which the output was calculated</td>
<td>Unordered set of unique space-separated tokens consisting of IDs*</td>
</tr>
<tr>
<td>form</td>
<td>button; fieldset; input; object; output; select; textarea; form-associated custom elements</td>
<td>Associates the element with a form element</td>
<td>ID*</td>
</tr>
<tr>
<td>formaction</td>
<td>button; input</td>
<td>URL to use for form submission</td>
<td>Valid non-empty URL potentially surrounded by spaces</td>
</tr>
<tr>
<td>formenctype</td>
<td>button; input</td>
<td>Entry list encoding type to use for form submission</td>
<td>&quot;text/plain&quot;, &quot;multipart/form-data&quot;, &quot;application/x-www-form-urlencoded&quot;</td>
</tr>
<tr>
<td>formmethod</td>
<td>button; input</td>
<td>Variant to use for form submission</td>
<td>&quot;GET&quot;, &quot;POST&quot;, &quot;dialog&quot;</td>
</tr>
<tr>
<td>formnovalidate</td>
<td>button; input</td>
<td>Bypass form control validation for form submission</td>
<td>Boolean attribute</td>
</tr>
<tr>
<td>formtarget</td>
<td>button</td>
<td>Browsing context for form submission</td>
<td>Valid browsing context name or keyword</td>
</tr>
<tr>
<td>headers</td>
<td>td; th</td>
<td>The header cells for this cell</td>
<td>Unordered set of unique space-separated tokens consisting of IDs*</td>
</tr>
<tr>
<td>height</td>
<td>canvas; embed; iframe; img; input; object; video</td>
<td>Vertical dimension</td>
<td>Valid non-negative integer</td>
</tr>
<tr>
<td>hidden</td>
<td>HTML elements</td>
<td>Whether the element is relevant</td>
<td>Boolean attribute</td>
</tr>
<tr>
<td>high</td>
<td>meter</td>
<td>Low limit of high range</td>
<td>Valid floating-point number</td>
</tr>
<tr>
<td>href</td>
<td>a; area</td>
<td>Address of the hyperlink</td>
<td>Valid URL potentially surrounded by spaces</td>
</tr>
<tr>
<td>href</td>
<td>link</td>
<td>Address of the hyperlink</td>
<td>Valid non-empty URL potentially surrounded by spaces</td>
</tr>
<tr>
<td>Attribute</td>
<td>Element(s)</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>href</td>
<td>base⁴⁷⁷</td>
<td>Document base URL⁴⁶⁹</td>
<td>Valid URL potentially surrounded by spaces⁴⁶⁹</td>
</tr>
<tr>
<td>hreflang</td>
<td>a²⁹⁸; link¹¹⁰</td>
<td>Language of the linked resource</td>
<td>Valid BCP 47 language tag</td>
</tr>
<tr>
<td>http-equiv</td>
<td>meta⁴⁷⁷</td>
<td>Pragma directive</td>
<td>&quot;content-type&quot;; &quot;default-style&quot;; &quot;refresh&quot;; &quot;x-ua-compatible&quot;; &quot;content-security-policy&quot;</td>
</tr>
<tr>
<td>id</td>
<td>HTML elements¹¹³⁸</td>
<td>The element's ID</td>
<td>Text¹¹³⁸</td>
</tr>
<tr>
<td>imagesizes</td>
<td>link¹¹⁰</td>
<td>Image sizes for different page layouts (for rel=“preload&quot;*⁴⁹⁸)</td>
<td>Valid source size list¹¹³⁶</td>
</tr>
<tr>
<td>imagesrcset</td>
<td>link¹¹⁰</td>
<td>Images to use in different situations, e.g., high-resolution displays, small monitors, etc. (for rel=“preload&quot;*⁴⁹⁸)</td>
<td>Comma-separated list of image candidate strings¹¹³⁶</td>
</tr>
<tr>
<td>inputmode</td>
<td>HTML elements⁷⁹⁵</td>
<td>Hint for selecting an input modality</td>
<td>&quot;none&quot;; &quot;text&quot;; &quot;tel&quot;; &quot;email&quot;; &quot;url&quot;; &quot;numeric&quot;; &quot;decimal&quot;; &quot;search&quot;</td>
</tr>
<tr>
<td>integrity</td>
<td>link¹¹⁰; script¹¹⁶</td>
<td>Integrity metadata used in Subresource Integrity checks</td>
<td>Text¹¹³⁰</td>
</tr>
<tr>
<td>is</td>
<td>HTML elements⁷⁰⁴</td>
<td>Creates a customized built-in element⁷⁰⁴</td>
<td>Valid custom element name⁷⁰⁵ of a defined customized built-in element⁷⁰⁴</td>
</tr>
<tr>
<td>ismap</td>
<td>img¹²⁺</td>
<td>Whether the image is a server-side image map</td>
<td>Boolean attribute⁷⁰⁷</td>
</tr>
<tr>
<td>itemid</td>
<td>HTML elements⁷¹⁷</td>
<td>Global identifier⁷¹⁷ for a microdata item</td>
<td>Valid URL potentially surrounded by spaces⁷¹⁸</td>
</tr>
<tr>
<td>itemprop</td>
<td>HTML elements⁷¹⁸</td>
<td>Property names⁷¹⁸ of a microdata item</td>
<td>Unordered set of unique space-separated tokens⁷¹⁸ consisting of valid absolute URLs, defined property names⁷¹⁸, or text*</td>
</tr>
<tr>
<td>itemref</td>
<td>HTML elements⁷¹⁷</td>
<td>Referenced⁷¹⁷ elements</td>
<td>Unordered set of unique space-separated tokens⁷¹⁸ consisting of IDs*</td>
</tr>
<tr>
<td>itemscope</td>
<td>HTML elements⁷¹⁶</td>
<td>Introduces a microdata item</td>
<td>Boolean attribute⁷⁰⁷</td>
</tr>
<tr>
<td>itemtype</td>
<td>HTML elements⁷⁷⁶</td>
<td>Item types⁷⁷⁶ of a microdata item</td>
<td>Unordered set of unique space-separated tokens⁷⁷⁶ consisting of valid absolute URLs*</td>
</tr>
<tr>
<td>kind</td>
<td>track⁵⁰⁸</td>
<td>The type of text track</td>
<td>&quot;subtitles&quot;; &quot;captions&quot;; &quot;descriptions&quot;; &quot;chapters&quot;; &quot;metadata&quot;</td>
</tr>
<tr>
<td>label</td>
<td>optgroup⁵⁴⁵; option⁵⁴⁶; track⁵⁰⁸</td>
<td>User-visible label</td>
<td>Text⁵⁰⁷</td>
</tr>
<tr>
<td>lang</td>
<td>HTML elements⁵¹⁴</td>
<td>Language⁵¹⁴ of the element</td>
<td>Valid BCP 47 language tag or the empty string</td>
</tr>
<tr>
<td>list</td>
<td>input⁵¹⁰</td>
<td>List of autocomplete options</td>
<td>ID*</td>
</tr>
<tr>
<td>loading</td>
<td>img¹²²; iframe¹³⁶</td>
<td>Used when determining loading deferment</td>
<td>&quot;lazy&quot;; &quot;eager&quot;</td>
</tr>
<tr>
<td>loop</td>
<td>audio⁹⁸⁵; video⁴¹⁰</td>
<td>Whether to loop the media resource⁴¹⁰</td>
<td>Boolean attribute⁴¹⁷</td>
</tr>
<tr>
<td>low</td>
<td>meter⁵⁵⁵</td>
<td>High limit of low range</td>
<td>Valid floating-point number⁵⁵⁵*</td>
</tr>
<tr>
<td>max</td>
<td>input⁵⁰⁶</td>
<td>Maximum value</td>
<td>Varies*</td>
</tr>
<tr>
<td>max</td>
<td>meter⁵⁵⁵; progress⁵⁵⁶</td>
<td>Upper bound of range</td>
<td>Valid floating-point number⁵⁵⁶*</td>
</tr>
<tr>
<td>maxlength</td>
<td>input⁴⁶⁷; textarea⁴⁷⁷</td>
<td>Maximum length of value</td>
<td>Valid non-negative integer⁴⁷⁷</td>
</tr>
<tr>
<td>media</td>
<td>link¹¹⁰; source¹²¹; style¹⁷⁶</td>
<td>Applicable media</td>
<td>Valid media query list¹⁷⁸</td>
</tr>
<tr>
<td>method</td>
<td>form⁶⁷¹</td>
<td>Variant to use for form submission⁶⁷⁹</td>
<td>&quot;GET&quot;; &quot;POST&quot;; &quot;dialog&quot;</td>
</tr>
<tr>
<td>min</td>
<td>input⁵⁰⁶</td>
<td>Minimum value</td>
<td>Varies*</td>
</tr>
<tr>
<td>min</td>
<td>meter⁵⁵⁵</td>
<td>Lower bound of range</td>
<td>Valid floating-point number⁵⁵⁵*</td>
</tr>
<tr>
<td>minlength</td>
<td>input⁴⁶⁷; textarea⁴⁷⁷</td>
<td>Minimum length of value</td>
<td>Valid non-negative integer⁴⁷⁷</td>
</tr>
<tr>
<td>multiple</td>
<td>input⁴⁶⁷; select⁴⁵⁹</td>
<td>Whether to allow multiple values</td>
<td>Boolean attribute⁴⁵⁹</td>
</tr>
<tr>
<td>muted</td>
<td>audio⁴⁸⁸; video⁴⁸⁸</td>
<td>Whether to mute the media resource⁴⁸⁸, by default</td>
<td>Boolean attribute⁴⁸⁸</td>
</tr>
<tr>
<td>name</td>
<td>button⁵⁰⁴; fieldset⁴⁸⁸; input⁴⁶⁸; output⁴⁶⁸; select⁴⁵⁹; textarea⁴⁷⁷; form-associated custom elements⁴⁹⁸</td>
<td>Name of the element to use for form submission⁴⁹⁸ and in the form elements⁴⁹⁸ API</td>
<td>Text⁴⁹⁸</td>
</tr>
<tr>
<td>name</td>
<td>form⁴⁹²</td>
<td>Name of form to use in the document.forms⁴⁹² API</td>
<td>Text⁴⁹²</td>
</tr>
<tr>
<td>name</td>
<td>iframe⁴³⁷; object⁴⁷⁵</td>
<td>Name of nested browsing</td>
<td>Valid browsing context name or keyword⁴²²</td>
</tr>
<tr>
<td>Attribute</td>
<td>Element(s)</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>name</td>
<td>meta[^56]</td>
<td>Metadata name</td>
<td>Text[^56]</td>
</tr>
<tr>
<td>name</td>
<td>param[^50]</td>
<td>Name of parameter</td>
<td>Text[^50]</td>
</tr>
<tr>
<td>name</td>
<td>slot[^50]</td>
<td>Name of shadow tree slot</td>
<td>Text[^50]</td>
</tr>
<tr>
<td>nomodule</td>
<td>script[^50]</td>
<td>Prevents execution in user agents that support `module scripts[^50]</td>
<td>Boolean attribute[^50]</td>
</tr>
<tr>
<td>nonce</td>
<td>HTML elements[^52]</td>
<td>Cryptographic nonce used in <code>Content Security Policy</code> checks (<code>CSP</code>)[^52]</td>
<td>Text[^52]</td>
</tr>
<tr>
<td>novalidate</td>
<td>form[^52]</td>
<td>Bypass form control validation for <code>form submission</code>[^52]</td>
<td>Boolean attribute[^52]</td>
</tr>
<tr>
<td>open</td>
<td>details[^54]</td>
<td>Whether the details are visible</td>
<td>Boolean attribute[^54]</td>
</tr>
<tr>
<td>open</td>
<td>dialog[^51]</td>
<td>Whether the dialog box is showing</td>
<td>Boolean attribute[^51]</td>
</tr>
<tr>
<td>optimum</td>
<td>meter[^53]</td>
<td>Optimum value in gauge</td>
<td>Valid floating-point number[^53]</td>
</tr>
<tr>
<td>pattern</td>
<td>input[^53]</td>
<td>Pattern to be matched by the form control's value</td>
<td>Regular expression matching the JavaScript <code>Pattern</code> production</td>
</tr>
<tr>
<td>ping</td>
<td>a[^50]; area[^25]</td>
<td>URLs to ping</td>
<td>Set of space-separated tokens[^50] consisting of `valid non-empty URLs[^50]</td>
</tr>
<tr>
<td>placeholder</td>
<td>input[^50]; textarea[^55]</td>
<td>User-visible label to be placed within the form control</td>
<td>Text[^50]</td>
</tr>
<tr>
<td>playsinline</td>
<td>video[^22]</td>
<td>Encourage the user agent to display video content within the element's playback area</td>
<td>Boolean attribute[^22]</td>
</tr>
<tr>
<td>poster</td>
<td>video[^21]</td>
<td>Poster frame to show prior to video playback</td>
<td>Valid non-empty URL potentially surrounded by spaces[^21]</td>
</tr>
<tr>
<td>preload</td>
<td>audio[^22]; video[^42]</td>
<td>Hints how much buffering the `media resource[^42] will likely need</td>
<td>&quot;none[^22]&quot;; &quot;metadata[^42]&quot;; &quot;auto[^22]&quot;</td>
</tr>
<tr>
<td>readonly</td>
<td>input[^22]; textarea[^58]</td>
<td>Whether to allow the value to be edited by the user</td>
<td>Boolean attribute[^58]</td>
</tr>
<tr>
<td>readonly</td>
<td>form-associat###[^58]</td>
<td>Affects `willValidate[^58] plus any behavior added by the custom element author</td>
<td>Boolean attribute[^58]</td>
</tr>
<tr>
<td>referrerpolicy</td>
<td>a[^50]; area[^25]; iframe[^25]; img[^25]; link[^25]; script[^25]</td>
<td>Referrer policy for fetches initiated by the element</td>
<td>Referrer policy</td>
</tr>
<tr>
<td>rel</td>
<td>a[^50]; area[^25]</td>
<td>Relationship between the location in the document containing the hyperlink[^25] and the destination resource</td>
<td>Unordered set of unique space-separated tokens[^50] *</td>
</tr>
<tr>
<td>rel</td>
<td>link[^59]</td>
<td>Relationship between the document containing the hyperlink[^59] and the destination resource</td>
<td>Unordered set of unique space-separated tokens[^59] *</td>
</tr>
<tr>
<td>required</td>
<td>input[^52]; select[^59]; textarea[^55]</td>
<td>Whether the control is required for `form submission[^55]</td>
<td>Boolean attribute[^59]</td>
</tr>
<tr>
<td>reversed</td>
<td>ol[^54]</td>
<td>Number the list backwards</td>
<td>Boolean attribute[^54]</td>
</tr>
<tr>
<td>rows</td>
<td>textarea[^55]</td>
<td>Number of lines to show</td>
<td>Valid non-negative integer[^55] greater than zero</td>
</tr>
<tr>
<td>rowspan</td>
<td>td[^46]; th[^46]</td>
<td>Number of rows that the cell is to span</td>
<td>Valid non-negative integer[^46]</td>
</tr>
<tr>
<td>sandbox</td>
<td>iframe[^34]</td>
<td>Security rules for nested content</td>
<td>Unordered set of unique space-separated tokens[^34], ASCII case-insensitive, consisting of &quot;allow-forms[^34]&quot;; &quot;allow-modals[^34]&quot;; &quot;allow-orientation-lock[^34]&quot;; &quot;allow-pointer-lock[^34]&quot;; &quot;allow-popups[^34]&quot;; &quot;allow-popups-to-escape-sandbox[^34]&quot;; &quot;allow-presentation[^34]&quot;; &quot;allow-same-origin[^34]&quot;; &quot;allow-scripts[^34]&quot; and &quot;allow-top-navigation[^34]&quot;</td>
</tr>
<tr>
<td>scope</td>
<td>th[^48]</td>
<td>Specifies which cells the header cell applies to</td>
<td>&quot;row[^48]&quot;; &quot;col[^48]&quot;; &quot;rowgroup[^48]&quot;; &quot;colgroup[^48]&quot;</td>
</tr>
<tr>
<td>selected</td>
<td>option[^57]</td>
<td>Whether the option is selected by default</td>
<td>Boolean attribute[^57]</td>
</tr>
<tr>
<td>shape</td>
<td>area[^50]</td>
<td>The kind of shape to be created</td>
<td>&quot;circle[^50]&quot;; &quot;default[^50]&quot;; &quot;poly[^50]&quot;; &quot;rect[^50]&quot;</td>
</tr>
<tr>
<td>Attribute</td>
<td>Element(s)</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>size</td>
<td><code>input</code>, <code>select</code></td>
<td>Size of the control</td>
<td>Valid non-negative integer greater than zero</td>
</tr>
<tr>
<td>sizes</td>
<td><code>link</code></td>
<td>Sizes of the icons (for <code>rel</code> and <code>icon</code>)</td>
<td>Unordered set of unique space-separated tokens, ASCII case-insensitive, consisting of sizes</td>
</tr>
<tr>
<td>sizes</td>
<td><code>img</code>, <code>source</code></td>
<td>Image sizes for different page layouts</td>
<td>Valid source size list</td>
</tr>
<tr>
<td>slot</td>
<td>HTML elements</td>
<td>The element's desired slot</td>
<td>Text</td>
</tr>
<tr>
<td>span</td>
<td><code>col</code>, <code>colgroup</code></td>
<td>Number of columns spanned by the element</td>
<td>Valid non-negative integer greater than zero</td>
</tr>
<tr>
<td>spellcheck</td>
<td>HTML elements</td>
<td>Whether the element is to have its spelling and grammar checked</td>
<td>&quot;true&quot;, &quot;false&quot;</td>
</tr>
<tr>
<td>src</td>
<td><code>audio</code>, <code>embed</code>, <code>iframe</code>, <code>img</code>, <code>input</code>, <code>script</code>, <code>source</code>, <code>track</code>, <code>video</code></td>
<td>Address of the resource</td>
<td>Valid non-empty URL potentially surrounded by spaces</td>
</tr>
<tr>
<td>srcdoc</td>
<td><code>iframe</code></td>
<td>A document to render in the <code>iframe</code></td>
<td>The source of an <code>iframe srcdoc</code> document</td>
</tr>
<tr>
<td>srclang</td>
<td><code>track</code></td>
<td>Language of the text track</td>
<td>Valid BCP 47 language tag</td>
</tr>
<tr>
<td>srcset</td>
<td><code>img</code>, <code>source</code></td>
<td>Images to use in different situations, e.g., high-resolution displays, small monitors, etc.</td>
<td>Comma-separated list of image candidate strings</td>
</tr>
<tr>
<td>start</td>
<td><code>ol</code></td>
<td>Starting value of the list</td>
<td>Valid integer</td>
</tr>
<tr>
<td>step</td>
<td><code>input</code></td>
<td>Granularity to be matched by the form control's value</td>
<td>Valid floating-point number greater than zero, or &quot;any&quot;</td>
</tr>
<tr>
<td>style</td>
<td>HTML elements</td>
<td>Presentational and formatting instructions</td>
<td>CSS declarations*</td>
</tr>
<tr>
<td>tabindex</td>
<td>HTML elements</td>
<td>Whether the element is focusable and sequentially focusable, and the relative order of the element for the purposes of sequential focus navigation</td>
<td>Valid integer</td>
</tr>
<tr>
<td>target</td>
<td><code>a</code>, <code>area</code></td>
<td>Browsing context for hyperlink traversal</td>
<td>Valid browsing context name or keyword</td>
</tr>
<tr>
<td>target</td>
<td><code>base</code></td>
<td>Default browsing context for hyperlink traversal</td>
<td>Valid browsing context name or keyword</td>
</tr>
<tr>
<td>target</td>
<td><code>form</code></td>
<td>Browsing context for form submission</td>
<td>Valid browsing context name or keyword</td>
</tr>
<tr>
<td>title</td>
<td>HTML elements</td>
<td>Advisory information for the element</td>
<td>Text</td>
</tr>
<tr>
<td>title</td>
<td><code>abbr</code>, <code>dfn</code></td>
<td>Full term or expansion of abbreviation</td>
<td>Text</td>
</tr>
<tr>
<td>title</td>
<td><code>input</code></td>
<td>Description of pattern (when used with <code>pattern</code> attribute)</td>
<td>Text</td>
</tr>
<tr>
<td>title</td>
<td><code>link</code></td>
<td>Title of the link</td>
<td>Text</td>
</tr>
<tr>
<td>title</td>
<td><code>link</code>, <code>style</code></td>
<td>CSS style sheet name</td>
<td>Text</td>
</tr>
<tr>
<td>translate</td>
<td>HTML elements</td>
<td>Whether the element is to be translated when the page is localized</td>
<td>&quot;yes&quot;, &quot;no&quot;</td>
</tr>
<tr>
<td>type</td>
<td><code>a</code>, <code>link</code></td>
<td>Hint for the type of the referenced resource</td>
<td>Valid MIME type string</td>
</tr>
<tr>
<td>type</td>
<td><code>button</code></td>
<td>Type of button</td>
<td>&quot;submit&quot;, &quot;reset&quot;, &quot;button&quot;</td>
</tr>
<tr>
<td>type</td>
<td><code>embed</code>, <code>object</code>, <code>source</code></td>
<td>Type of embedded resource</td>
<td>Valid MIME type string</td>
</tr>
<tr>
<td>type</td>
<td><code>input</code></td>
<td>Type of form control</td>
<td>Input type keyword</td>
</tr>
<tr>
<td>type</td>
<td><code>ol</code></td>
<td>Kind of list marker</td>
<td>1, 2, 3, a, A, 1.1, 2.2, A.1, 1.2, 2.3, 3.1</td>
</tr>
<tr>
<td>type</td>
<td><code>script</code></td>
<td>Type of script</td>
<td>&quot;module&quot;, a valid MIME type string that is not a JavaScript MIME type essence match</td>
</tr>
<tr>
<td>usemap</td>
<td><code>img</code>, <code>object</code></td>
<td>Name of <code>image map</code> to use</td>
<td>Valid hash-name reference</td>
</tr>
<tr>
<td>value</td>
<td><code>button</code>, <code>option</code></td>
<td>Value to be used for <code>form submission</code></td>
<td>Text</td>
</tr>
</tbody>
</table>
An asterisk (*) in a cell indicates that the actual rules are more complicated than indicated in the table above.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Element(s)</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>onloadedmetadata</td>
<td>HTML elements</td>
<td>loadedmetadata event handler</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onloadstart</td>
<td>HTML elements</td>
<td>loadstart event handler</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onmessage</td>
<td>body</td>
<td>message event handler for Window object</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onmessageerror</td>
<td>body</td>
<td>messageerror event handler for Window object</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onmousedown</td>
<td>HTML elements</td>
<td>mousedown event handler</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onmouseout</td>
<td>HTML elements</td>
<td>mouseout event handler</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onmousemove</td>
<td>HTML elements</td>
<td>mousemove event handler</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onmouseenter</td>
<td>HTML elements</td>
<td>mouseenter event handler</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onmouseleave</td>
<td>HTML elements</td>
<td>mouseleave event handler</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onmousemove</td>
<td>body</td>
<td>mousemove event handler</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onmouseout</td>
<td>body</td>
<td>mouseout event handler</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onmousewheel</td>
<td>body</td>
<td>mousewheel event handler</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onvolumechange</td>
<td>body</td>
<td>volumechange event handler</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onunload</td>
<td>body</td>
<td>unload event handler</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onunhandledrejection</td>
<td>body</td>
<td>unhandledrejection event handler for Window object</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onresize</td>
<td>body</td>
<td>resize event handler</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onscroll</td>
<td>body</td>
<td>scroll event handler</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onsubmit</td>
<td>body</td>
<td>submit event handler</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onsuspend</td>
<td>body</td>
<td>suspend event handler</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>ontimeupdate</td>
<td>body</td>
<td>timeupdate event handler</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>ontoggle</td>
<td>body</td>
<td>toggle event handler</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onunhandledrejection</td>
<td>body</td>
<td>unhandledrejection event handler for Window object</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onunload</td>
<td>body</td>
<td>unload event handler</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onvolumechange</td>
<td>body</td>
<td>volumechange event handler</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onwaiting</td>
<td>body</td>
<td>waiting event handler</td>
<td>Event handler content attribute</td>
</tr>
<tr>
<td>onwheel</td>
<td>body</td>
<td>wheel event handler</td>
<td>Event handler content attribute</td>
</tr>
</tbody>
</table>

**Element Interfaces**

This section is non-normative.

<table>
<thead>
<tr>
<th>Element(s)</th>
<th>Interface(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>HTMLAnchorElement - HTMLElement</td>
</tr>
<tr>
<td>abbr</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>address</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>area</td>
<td>HTMLAreaElement - HTMLElement</td>
</tr>
<tr>
<td>Element(s)</td>
<td>Interface(s)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
</tr>
<tr>
<td>article</td>
<td>HTML Element</td>
</tr>
<tr>
<td>aside</td>
<td>HTML Element</td>
</tr>
<tr>
<td>audio</td>
<td>HTMLAudioElement, HTMLMediaElement, HTMLElement</td>
</tr>
<tr>
<td>b</td>
<td>HTML Element</td>
</tr>
<tr>
<td>base</td>
<td>HTMLBaseElement, HTMLElement</td>
</tr>
<tr>
<td>bdo</td>
<td>HTML Element</td>
</tr>
<tr>
<td>blockquote</td>
<td>HTMLQuoteElement, HTMLElement</td>
</tr>
<tr>
<td>body</td>
<td>HTMLBodyElement, HTMLElement</td>
</tr>
<tr>
<td>br</td>
<td>HTMLBRElement, HTML Element</td>
</tr>
<tr>
<td>button</td>
<td>HTMLButtonElement, HTMLElement</td>
</tr>
<tr>
<td>canvas</td>
<td>HTMLCanvasElement, HTMLElement</td>
</tr>
<tr>
<td>caption</td>
<td>HTMLTableCaptionElement, HTMLElement</td>
</tr>
<tr>
<td>cite</td>
<td>HTML Element</td>
</tr>
<tr>
<td>code</td>
<td>HTML Element</td>
</tr>
<tr>
<td>col</td>
<td>HTMLTableColElement, HTMLElement</td>
</tr>
<tr>
<td>colgroup</td>
<td>HTMLTableColElement, HTMLElement</td>
</tr>
<tr>
<td>data</td>
<td>HTMLDataElement, HTMLElement</td>
</tr>
<tr>
<td>datalist</td>
<td>HTMLDataListElement, HTMLElement</td>
</tr>
<tr>
<td>dd</td>
<td>HTML Element</td>
</tr>
<tr>
<td>del</td>
<td>HTMLModElement, HTMLElement</td>
</tr>
<tr>
<td>details</td>
<td>HTMLDetailsElement, HTMLElement</td>
</tr>
<tr>
<td>dfn</td>
<td>HTML Element</td>
</tr>
<tr>
<td>dialog</td>
<td>HTMLDialogElement, HTMLElement</td>
</tr>
<tr>
<td>div</td>
<td>HTMLDivElement, HTMLElement</td>
</tr>
<tr>
<td>dl</td>
<td>HTMLDListElement, HTMLElement</td>
</tr>
<tr>
<td>dt</td>
<td>HTML Element</td>
</tr>
<tr>
<td>em</td>
<td>HTML Element</td>
</tr>
<tr>
<td>embed</td>
<td>HTMLEmbedElement, HTMLElement</td>
</tr>
<tr>
<td>fieldset</td>
<td>HTMLFieldSetElement, HTMLElement</td>
</tr>
<tr>
<td>figcaption</td>
<td>HTML Element</td>
</tr>
<tr>
<td>figure</td>
<td>HTML Element</td>
</tr>
<tr>
<td>footer</td>
<td>HTML Element</td>
</tr>
<tr>
<td>form</td>
<td>HTMLFormElement, HTMLElement</td>
</tr>
<tr>
<td>h1</td>
<td>HTMLHeadingElement, HTMLElement</td>
</tr>
<tr>
<td>h2</td>
<td>HTMLHeadingElement, HTMLElement</td>
</tr>
<tr>
<td>h3</td>
<td>HTMLHeadingElement, HTMLElement</td>
</tr>
<tr>
<td>h4</td>
<td>HTMLHeadingElement, HTMLElement</td>
</tr>
<tr>
<td>h5</td>
<td>HTMLHeadingElement, HTMLElement</td>
</tr>
<tr>
<td>h6</td>
<td>HTMLHeadingElement, HTMLElement</td>
</tr>
<tr>
<td>head</td>
<td>HTMLHeadElement, HTMLElement</td>
</tr>
<tr>
<td>header</td>
<td>HTML Element</td>
</tr>
<tr>
<td>hgroup</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>hr</td>
<td>HTMLHRElement, HTMLElement</td>
</tr>
<tr>
<td>html</td>
<td>HTMLHtmlElement, HTMLElement</td>
</tr>
<tr>
<td>i</td>
<td>HTML Element</td>
</tr>
<tr>
<td>iframe</td>
<td>HTMLIFrameElement, HTMLElement</td>
</tr>
<tr>
<td>img</td>
<td>HTMLImageElement, HTMLElement</td>
</tr>
<tr>
<td>input</td>
<td>HTMLInputElement, HTMLElement</td>
</tr>
<tr>
<td>ins</td>
<td>HTMLModElement, HTMLElement</td>
</tr>
<tr>
<td>kbd</td>
<td>HTML Element</td>
</tr>
<tr>
<td>label</td>
<td>HTMLLabelElement, HTMLElement</td>
</tr>
<tr>
<td>legend</td>
<td>HTMLLegendElement, HTMLElement</td>
</tr>
<tr>
<td>li</td>
<td>HTMLLIElement, HTMLElement</td>
</tr>
<tr>
<td>Element(s)</td>
<td>Interface(s)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
</tr>
<tr>
<td>link</td>
<td>HTMLLinkElement : HTMLElement</td>
</tr>
<tr>
<td>main</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>map</td>
<td>HTMLMapElement : HTMLElement</td>
</tr>
<tr>
<td>mark</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>menu</td>
<td>HTMLMenuElement : HTMLElement</td>
</tr>
<tr>
<td>meta</td>
<td>HTMLMetaElement : HTMLElement</td>
</tr>
<tr>
<td>meter</td>
<td>HTMLMeterElement : HTMLElement</td>
</tr>
<tr>
<td>nav</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>noscript</td>
<td>HTMLDivElement</td>
</tr>
<tr>
<td>object</td>
<td>HTMLObjectElement : HTMLElement</td>
</tr>
<tr>
<td>ol</td>
<td>HTMLUListElement : HTMLElement</td>
</tr>
<tr>
<td>optgroup</td>
<td>HTMLUListElement : HTMLElement</td>
</tr>
<tr>
<td>option</td>
<td>HTMLOptionElement : HTMLElement</td>
</tr>
<tr>
<td>output</td>
<td>HTMLOutputElement : HTMLElement</td>
</tr>
<tr>
<td>p</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>param</td>
<td>HTMLParagraphElement : HTMLElement</td>
</tr>
<tr>
<td>picture</td>
<td>HTMLImageElement : HTMLElement</td>
</tr>
<tr>
<td>pre</td>
<td>HTMLPreElement : HTMLElement</td>
</tr>
<tr>
<td>progress</td>
<td>HTMLProgressElement : HTMLElement</td>
</tr>
<tr>
<td>q</td>
<td>HTMQLiteralElement : HTMLElement</td>
</tr>
<tr>
<td>rp</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>rt</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>ruby</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>s</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>samp</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>script</td>
<td>HTMLScriptElement : HTMLElement</td>
</tr>
<tr>
<td>section</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>select</td>
<td>HTMLSelectElement : HTMLElement</td>
</tr>
<tr>
<td>slot</td>
<td>HTMLOutputElement : HTMLElement</td>
</tr>
<tr>
<td>small</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>source</td>
<td>HTMLSourceElement : HTMLElement</td>
</tr>
<tr>
<td>span</td>
<td>HTMLSpanElement : HTMLElement</td>
</tr>
<tr>
<td>strong</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>style</td>
<td>HTMLStyleElement : HTMLElement</td>
</tr>
<tr>
<td>sub</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>summary</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>sup</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>table</td>
<td>HTMLOutputElement : HTMLElement</td>
</tr>
<tr>
<td>tbody</td>
<td>HTMLTableSectionElement : HTMLElement</td>
</tr>
<tr>
<td>td</td>
<td>HTMLTableCellElement : HTMLElement</td>
</tr>
<tr>
<td>template</td>
<td>HTMLTemplateElement : HTMLElement</td>
</tr>
<tr>
<td>textarea</td>
<td>HTMLTextAreaElement : HTMLElement</td>
</tr>
<tr>
<td>tfoot</td>
<td>HTMLTableSectionElement : HTMLElement</td>
</tr>
<tr>
<td>th</td>
<td>HTMLTableCellElement : HTMLElement</td>
</tr>
<tr>
<td>thead</td>
<td>HTMLTableSectionElement : HTMLElement</td>
</tr>
<tr>
<td>time</td>
<td>HTMLOutputElement : HTMLElement</td>
</tr>
<tr>
<td>title</td>
<td>HTMLTitleElement : HTMLElement</td>
</tr>
<tr>
<td>track</td>
<td>HTMLTrackElement : HTMLElement</td>
</tr>
<tr>
<td>track</td>
<td>HTMLTrackElement : HTMLElement</td>
</tr>
<tr>
<td>ul</td>
<td>HTMLUListElement : HTMLElement</td>
</tr>
<tr>
<td>var</td>
<td>HTMLElement</td>
</tr>
<tr>
<td>video</td>
<td>HTMLVideoElement : HTMLMediaElement : HTMLElement</td>
</tr>
<tr>
<td>wbr</td>
<td>HTMLElement</td>
</tr>
</tbody>
</table>
### All Interfaces

This section is non-normative.

- `AudioTrack`\(^{p418}\)
- `AudioTrackList`\(^{p418}\)
- `BarProp`\(^{p834}\)
- `BeforeUnloadEvent`\(^{p896}\)
- `BroadcastChannel`\(^{p1008}\)
- `CanvasGradient`\(^{p641}\)
- `CanvasPattern`\(^{p642}\)
- `CanvasRenderingContext2D`\(^{p638}\)
- `CloseEvent`\(^{p997}\)
- `CustomElementRegistry`\(^{p707}\)
- `DOMParser`\(^{p958}\)
- `DOMStringList`\(^{p102}\)
- `DOMStringMap`\(^{p147}\)
- `DataTransfer`\(^{p799}\)
- `DataTransferItem`\(^{p803}\)
- `DataTransferItemList`\(^{p802}\)
- `DedicatedWorkerGlobalScope`\(^{p1029}\)
- `Document`\(^{p115}\)
- `DragEvent`\(^{p804}\)
- `ElementInternals`\(^{p716}\)
- `ErrorEvent`\(^{p923}\)
- `EventSource`\(^{p983}\)
- `External`\(^{p1234}\)
- `FormDataEvent`\(^{p603}\)
- `HTMLAllCollection`\(^{p97}\)
- `HTMLAnchorElement`\(^{p1225}\)
- `HTMLAreaElement`\(^{p1226}\)
- `HTMLAudioElement`\(^{p385}\)
- `HTMLBRElement`\(^{p1226}\)
- `HTMLBaseElement`\(^{p156}\)
- `HTMLBodyElement`\(^{p180}\)
- `HTMLButtonElement`\(^{p536}\)
- `HTMLCanvasElement`\(^{p634}\)
- `HTMLDListElement`\(^{p1227}\)
- `HTMLDataElement`\(^{p981}\)
- `HTMLDataListElement`\(^{p543}\)
- `HTMLDetailsElement`\(^{p1227}\)
- `HTMLDialogElement`\(^{p611}\)
- `HTMLDirectoryElement`\(^{p1227}\)
- `HTMLDivElement`\(^{p239}\)
- `HTMLElement`\(^{p125}\)
- `HTMLEmbedElement`\(^{p370}\)
- `HTMLFieldSetElement`\(^{p562}\)
- `HTMLFontElement`\(^{p1227}\)
- `HTMLFormControlsCollection`\(^{p99}\)
- `HTMLFormElement`\(^{p487}\)
- `HTMLIFrameElement`\(^{p362}\)
- `HTMLImageElement`\(^{p321}\)
- `HTMLInputElement`\(^{p494}\)
- `HTMLLIElement`\(^{p226}\)
- `HTMLLabelElement`\(^{p491}\)
- `HTMLLegendElement`\(^{p565}\)
- `HTMLLinkElement`\(^{p158}\)
- `HTMLMapElement`\(^{p166}\)
- `HTMLMarqueeElement`\(^{p1222}\)
- `HTMLMediaElement`\(^{p188}\)
- `HTMLOptionElement`\(^{p1230}\)
- `HTMLParagraphElement`\(^{p1228}\)
### Events

This section is non-normative.

<table>
<thead>
<tr>
<th>Event</th>
<th>Interface</th>
<th>Interesting targets</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abort</td>
<td>Event</td>
<td>Window[^827]</td>
<td>Fired at the Window[^827] when the download was aborted by the user</td>
</tr>
<tr>
<td>DOMContentLoaded</td>
<td>Event</td>
<td>Document[^111]</td>
<td>Fired at the Document[^111] once the parser has finished</td>
</tr>
<tr>
<td>afterprint</td>
<td>Event</td>
<td>Window[^827]</td>
<td>Fired at the Window[^827] after printing</td>
</tr>
<tr>
<td>beforeprint</td>
<td>Event</td>
<td>Window[^827]</td>
<td>Fired at the Window[^827] before printing</td>
</tr>
<tr>
<td>beforeunload</td>
<td>BeforeUnloadEvent[^806]</td>
<td>Window[^827]</td>
<td>Fired at the Window[^827] when the page is about to be unloaded, in case the page would like to show a warning prompt</td>
</tr>
<tr>
<td>blur</td>
<td>Event</td>
<td>Window[^827] elements</td>
<td>Fired at nodes when they stop being focused[^773]</td>
</tr>
<tr>
<td>cancel</td>
<td>Event</td>
<td>dialog[^816] elements</td>
<td>Fired at dialog[^816] elements when they are canceled by the user (e.g. by pressing the Escape key)</td>
</tr>
<tr>
<td>change</td>
<td>Event</td>
<td>Form controls</td>
<td>Fired at controls when the user commits a value change (see also the input[^1266] event)</td>
</tr>
<tr>
<td>click</td>
<td>PointerEvent</td>
<td>Elements</td>
<td>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)</td>
</tr>
<tr>
<td>close</td>
<td>Event or CloseEvent[^907]</td>
<td>dialog[^816] elements, WebSocket[^983]</td>
<td>Fired at dialog[^816] elements when they are closed, and at WebSocket[^983] objects when the connection is terminated</td>
</tr>
<tr>
<td>connect</td>
<td>MessageEvent[^981]</td>
<td>SharedWorkerGlobalScope[^980]</td>
<td>Fired at a shared worker's global scope when a new client connects</td>
</tr>
<tr>
<td>contextmenu</td>
<td>PointerEvent</td>
<td>Elements</td>
<td>Fired at elements when the user requests their context menu</td>
</tr>
<tr>
<td>copy</td>
<td>Event</td>
<td>Elements</td>
<td>Fired at elements when the user copies data to the clipboard</td>
</tr>
<tr>
<td>cut</td>
<td>Event</td>
<td>Elements</td>
<td>Fired at elements when the user copies the selected data on the clipboard and removes the selection from the document</td>
</tr>
<tr>
<td>error</td>
<td>Event or ErrorEvent[^923]</td>
<td>Global scope objects, Worker[^1089] objects, elements, networking-related objects</td>
<td>Fired when unexpected errors occur (e.g. networking errors, script errors, decoding errors)</td>
</tr>
<tr>
<td>focus</td>
<td>Event</td>
<td>Window[^827] elements</td>
<td>Fired at nodes gaining focus[^777]</td>
</tr>
<tr>
<td>formdata</td>
<td>FormDataEvent[^603]</td>
<td>form[^804] elements</td>
<td>Fired at a form[^804] element when it is constructing the entry list[^604]</td>
</tr>
<tr>
<td>hashchange</td>
<td>HashChangeEvent[^383]</td>
<td>Window[^827]</td>
<td>Fired at the Window[^827] when the fragment part of the document's URL changes</td>
</tr>
<tr>
<td>input</td>
<td>Event</td>
<td>Form controls</td>
<td>Fired at controls when the user changes the value (see also the change[^1266] event)</td>
</tr>
<tr>
<td>invalid</td>
<td>Event</td>
<td>Form controls</td>
<td>Fired at controls during form validation if they do not satisfy their constraints</td>
</tr>
<tr>
<td>languagechange</td>
<td>Event</td>
<td>Global scope objects</td>
<td>Fired at the global scope object when the user's preferred languages change</td>
</tr>
<tr>
<td>load</td>
<td>Event</td>
<td>Window[^827] elements</td>
<td>Fired at the Window[^827] when the document has finished loading; fired at an element containing a resource (e.g. img[^131], embed[^260]) when its resource has finished loading</td>
</tr>
<tr>
<td>message</td>
<td>MessageEvent[^981]</td>
<td>Window[^827], EventSource[^981], WebSocket[^981], MessagePort[^135], BroadcastChannel[^3983], DedicatedWorkerGlobalScope[^825], Worker[^1035], ServiceWorkerContainer</td>
<td>Fired at an object when it receives a message</td>
</tr>
<tr>
<td>messageerror</td>
<td>MessageEvent[^981]</td>
<td>Window[^827], MessagePort[^1065]</td>
<td>Fired at an object when it receives a message that</td>
</tr>
<tr>
<td>Event</td>
<td>Interface</td>
<td>Interesting targets</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>offline</td>
<td>Event</td>
<td>Global scope objects</td>
<td>Fired at the global scope object when the network connections fail</td>
</tr>
<tr>
<td>online</td>
<td>Event</td>
<td>Global scope objects</td>
<td>Fired at the global scope object when the network connections returns</td>
</tr>
<tr>
<td>open</td>
<td>Event</td>
<td>EventSource[^11], WebSocket[^11]</td>
<td>Fired at networking-related objects when a connection is established</td>
</tr>
<tr>
<td>pagehide</td>
<td>PageTransitionEvent[^11]</td>
<td>Window[^27]</td>
<td>Fired at the Window[^27] when the page's entry in the session history[^11] stops being the current entry[^27]</td>
</tr>
<tr>
<td>pageshow</td>
<td>PageTransitionEvent[^11]</td>
<td>Window[^27]</td>
<td>Fired at the Window[^27] when the page's entry in the session history[^11] becomes the current entry[^27]</td>
</tr>
<tr>
<td>paste</td>
<td>Event</td>
<td>Elements</td>
<td>Fired at elements when the user will insert the clipboard data in the most suitable format (if any) supported for the given context</td>
</tr>
<tr>
<td>popstate</td>
<td>PopStateEvent[^11]</td>
<td>Window[^27]</td>
<td>Fired at the Window[^27] when the user navigates the session history[^11]</td>
</tr>
<tr>
<td>readystatechange</td>
<td>Event</td>
<td>Document[^11]</td>
<td>Fired at the Document[^11] when it finishes parsing and again when all its subresources have finished loading</td>
</tr>
<tr>
<td>rejectionhandled</td>
<td>PromiseRejectionEvent[^20]</td>
<td>Global scope objects</td>
<td>Fired at global scope objects when a previously-unhandled promise rejection becomes handled</td>
</tr>
<tr>
<td>reset</td>
<td>Event</td>
<td>form[^10] elements</td>
<td>Fired at a form[^10] element when it is reset[^10]</td>
</tr>
<tr>
<td>securitypolicyviolation</td>
<td>Event</td>
<td>Elements</td>
<td>Fired at elements when a Content Security Policy violation is generated [CSP][^12]</td>
</tr>
<tr>
<td>select</td>
<td>Event</td>
<td>Form controls</td>
<td>Fired at form controls when their text selection is adjusted (whether by an API or by the user)</td>
</tr>
<tr>
<td>slotchange</td>
<td>Event</td>
<td>slot[^10] elements</td>
<td>Fired at slot[^10] elements when their assigned nodes change</td>
</tr>
<tr>
<td>storage</td>
<td>StorageEvent[^12]</td>
<td>Window[^27]</td>
<td>Fired at Window[^27] event when the corresponding localStorage[^12] or sessionStorage[^12] storage areas change</td>
</tr>
<tr>
<td>submit</td>
<td>Event</td>
<td>form[^10] elements</td>
<td>Fired at a form[^10] element when it is submitted[^10]</td>
</tr>
<tr>
<td>toggle</td>
<td>Event</td>
<td>details[^10] element</td>
<td>Fired at details[^10] elements when they open or close</td>
</tr>
<tr>
<td>unhandledrejection</td>
<td>PromiseRejectionEvent[^20]</td>
<td>Global scope objects</td>
<td>Fired at global scope objects when a promise rejection goes unhandled</td>
</tr>
<tr>
<td>unload</td>
<td>Event</td>
<td>Window[^27]</td>
<td>Fired at the Window[^27] object when the page is going away</td>
</tr>
</tbody>
</table>


**MIME Types** \[^5\] 67

This section is non-normative.

The following MIME types are mentioned in this specification:

- application/atom+xml
  - Atom [ATOM]\[^1\]
- application/ecmascript
  - JavaScript (legacy type) [JAVASCRIPT]\[^2\]
- application/javascript
  - JavaScript (legacy type) [JAVASCRIPT]\[^2\]
- application/json
  - JSON [JSON]\[^2\]
text/javascript1.4
  JavaScript (legacy type) [JAVASCRIPT]^{1273}

text/javascript1.5
  JavaScript (legacy type) [JAVASCRIPT]^{1273}

text/jscript
  JavaScript (legacy type) [JAVASCRIPT]^{1273}

text/json
  JSON (legacy type)

text/livescript
  JavaScript (legacy type) [JAVASCRIPT]^{1273}

text/plain
  Generic plain text [RFC2046]^{1275} [RFC3676]^{1275}

text/html^{1276}
  HTML

text/ping^{1279}
  Hyperlink auditing

text/uri-list
  List of URLs [RFC2483]^{1275}

text/vcard
  vCard [RFC6350]^{1276}

text/vtt
  WebVTT [WEBVTT]^{1277}

text/x-ecmascript
  JavaScript (legacy type) [JAVASCRIPT]^{1273}

text/x-javascript
  JavaScript (legacy type) [JAVASCRIPT]^{1273}

text/xml
  XML [XML]^{1278} [RFC7303]^{1276}

video/mp4
  MPEG-4 video [RFC4337]^{1276}

video/mpeg
  MPEG video [RFC2046]^{1275}


References

All references are normative unless marked "Non-normative".

[ABNF]  
*Augmented BNF for Syntax Specifications: ABNF*, D. Crocker, P. Overell. IETF.

[ABOUT]  
*The 'about' URI scheme*, S. Moonesamy. IETF.

[APNG]  

[ARIA]  

[ARIAHTML]  
*ARIA in HTML*, S. Faulkner, S. O'Hara. W3C.

[ATAG]  

[ATOM]  
(Non-normative) *The Atom Syndication Format*, M. Nottingham, R. Sayre. IETF.

[BATTERY]  
(Non-normative) *Battery Status API*, A. Kostiainen, M. Lamouri. W3C.

[BCP47]  
*Tags for Identifying Languages; Matching of Language Tags*, A. Phillips, M. Davis. IETF.

[BEZIER]  

[BIDI]  

[BOCU1]  

[CESU8]  
(Non-normative) *UTR #26: Compatibility Encoding Scheme For UTF-16: 8-BIT (CESU-8)*, T. Phipps. Unicode Consortium.

[CHARMOD]  

[COMPOSITE]  
*Compositing and Blending*, R. Cabanier, N. Andronikos. W3C.

[COMPUTABLE]  

[COOKIES]  
*HTTP State Management Mechanism*, A. Barth. IETF.

[CSP]  
*Content Security Policy*, M. West, D. Veditz. W3C.

[CSS]  
*Cascading Style Sheets Level 2 Revision 2*, B. Bos, T. Çelik, I. Hickson, H. Lie. W3C.

[CSSALIGN]  
*CSS Box Alignment*, E. Etemad, T. Atkins. W3C.

CSS Style Attributes, Ç. Çelik, E. Etemad. W3C.

CSS Backgrounds and Borders, B. Bos, E. Etemad, B. Kemper. W3C.

CSS Box Model, E. Etemad. W3C.

CSS Cascading and Inheritance, E. Etemad, T. Atkins. W3C.

CSS Color Module, Ç. Çelik, C. Lilley, L. Baron. W3C.

CSS Color Adjustment Module, E. Etemad, R. Atanassov, R. Lillesveen, T. Atkins. W3C.

CSS Display, T. Atkins, E. Etemad. W3C.

CSS Font Loading, T. Atkins, J. Daggett. W3C.

CSS Fonts, J. Daggett. W3C.

CSS Flexible Box Layout, T. Atkins, E. Etemad, R. Atanassov. W3C.

CSS Generated Content, H. Lie, E. Etemad, I. Hickson. W3C.

CSS Grid Layout, T. Atkins, E. Etemad, R. Atanassov. W3C.

CSS Images Module, E. Etemad, T. Atkins, L. Verou. W3C.

CSS Images Module Level 4, E. Etemad, T. Atkins, L. Verou. W3C.

CSS Inline Layout, D. Cramer, E. Etemad. W3C.

CSS Lists and Counters, T. Atkins. W3C.

CSS Logical Properties, R. Atanassov, E. Etemad. W3C.

CSS Multi-column Layout, H. Lie, F. Rivoal, R. Andrew. W3C.

Cascading Style Sheets Object Model (CSSOM), S. Pieters, G. Adams. W3C.

CSSOM View Module, S. Pieters, G. Adams. W3C.

CSS Overflow Module, L. Baron, F. Rivoal. W3C.

CSS Painting API, I. Kilpatrick, D. Jackson. W3C.
[CSSPOSITION]
  CSS Positioned Layout, R. Atanassov, A. Eicholz. W3C.

[CSSPSEUDO]
  CSS Pseudo-Elements, D. Glazman, E. Etemad, A. Stearns. W3C.

[CSSRUBY]
  CSS3 Ruby Module, R. Ishida. W3C.

[CSSSCOPING]
  CSS Scoping Module, T. Atkins. W3C.

[CSSSIZING]
  CSS Intrinsic & Extrinsic Sizing Module, T. Atkins, E. Etemad. W3C.

[CSSTRANSITIONS]
  (Non-normative) CSS Transitions, D. Jackson, D. Hyatt, C. Marrin, L. Baron. W3C.

[CSSUI]
  CSS3 Basic User Interface Module, T. Çelik. W3C.

[CSSSYNTAX]
  CSS Syntax, T. Atkins, S. Sapin. W3C.

[CSSTABLE]
  CSS Table, F. Remy, G. Whitworth. W3C.

[CSSTEXT]
  CSS Text, E. Etemad, K. Ishii. W3C.

[CSSVALUES]
  CSS3 Values and Units, H. Lie, T. Atkins, E. Etemad. W3C.

[CSSWM]
  CSS Writing Modes, E. Etemad, K. Ishii. W3C.

[DASH]
  Dynamic adaptive streaming over HTTP (DASH), ISO.

[DOM]
  DOM, A. van Kesteren, A. Gregor, Ms2ger. WHATWG.

[DOMPARSING]
  DOM Parsing and Serialization, T. Leithead. W3C.

[DOT]
  (Non-normative) The DOT Language, Graphviz.

[E163]

[ENCODING]
  Encoding, A. van Kesteren, J. Bell. WHATWG.

[EXECCOMMAND]
  execCommand, J. Wilm, A. Gregor. W3C Editing APIs CG.

[EXIF]
  (Non-normative) Exchangeable image file format, JEITA.

[FETCH]
  Fetch, A. van Kesteren. WHATWG.

[FILEAPI]
  File API, A. Ranganathan. W3C.

[FILTERS]
Fullscreen, A. van Kesteren, T. Çelik. WHATWG.

Geometry Interfaces, S. Pieters, D. Schulze, R. Cabanier. W3C.

Graphics Interchange Format, CompuServe.


High Resolution Time, I. Grigorik, J. Simonsen, J. Mann. W3C.

HTML Accessibility API Mappings 1.0, S. Faulkner, A. Surkov, S. O'Hara. W3C.

Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing, R. Fielding, J. Reschke. IETF.
Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, R. Fielding, J. Reschke. IETF.
Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests, R. Fielding, J. Reschke. IETF.
Hypertext Transfer Protocol (HTTP/1.1): Range Requests, R. Fielding, Y. Lafon, J. Reschke. IETF.
Hypertext Transfer Protocol (HTTP/1.1): Caching, R. Fielding, M. Nottingham, J. Reschke. IETF.
Hypertext Transfer Protocol (HTTP/1.1): Authentication, R. Fielding, J. Reschke. IETF.

Indexed Database API, A. Alabbas, J. Bell. W3C.

Sourcing In-band Media Resource Tracks from Media Containers into HTML, S. Pfeiffer, B. Lund. W3C.

Infra, A. van Kesteren, D. Denicola. WHATWG.

Intersection Observer, S. Zager. W3C.

ISO 3166: Codes for the representation of names of countries and their subdivisions, ISO.

ISO 4217: Codes for the representation of currencies and funds, ISO.

ISO 8601: Data elements and interchange formats — Information interchange — Representation of dates and times, ISO.


Requirements for Japanese Text Layout, W3C.

JPEG File Interchange Format, E. Hamilton.

Error Stacks, Ecma International.

ECMAScript Internationalization API Specification, Ecma International.

Top-Level Await, Ecma International.
The JavaScript Object Notation (JSON) Data Interchange Format, T. Bray. IETF.

Long Tasks, D. Denicola, I. Grigorik, S. Panicker. W3C.

(Non-normative) The 'mailto' URI scheme, M. Duerst, L. Masinter, J. Zawinski. IETF.


Mathematical Markup Language (MathML), D. Carlisle, P. Ion, R. Miner. W3C.

Media Fragments URI, R. Troncy, E. Mannens, S. Pfeiffer, D. Van Deursen. W3C.

Media Source Extensions, A. Colwell, A. Bateman, M. Watson. W3C.

Media Capture and Streams, D. Burnett, A. Bergkvist, C. Jennings, A. Narayanan. W3C.

Reporting, D. Creager, I. Clelland, M. West. W3C.

Mixed Content, M. West. W3C.

MNG (Multiple-image Network Graphics) Format, G. Randers-Pehrson.


ISO/IEC 14496-12: ISO base media file format. ISO/IEC.

Media Queries, H. Lie, T. Çelik, D. Glazman, A. van Kesteren. W3C.

(Non-normative) Multiple buffering. Wikipedia.


(Non-normative) Gecko Plugin API Reference. Mozilla.

SkeletonHeaders, Xiph.Org.

Autodiscovery in HTML/XHTML, In OpenSearch 1.1 Draft 4, Section 4.6.2. OpenSearch.org.

The Web Origin Concept, A. Barth. IETF.

Paint Timing, S. Panicker. W3C.
Page Visibility Level 2, I. Grigorik, A. Jain, J. Mann. W3C.

Payment Request API, M. Cáceres, D. Wang, R. Solomakhin, I. Jacobs. W3C.

Document management — Portable document format — Part 1: PDF, ISO.

Permissions Policy, I. Clelland, W3C.

Pingback 1.0, S. Langridge, I. Hickson.


Pointer Events, J. Rossi, M. Brubeck, R. Byers, P. H. Lauke. W3C.

Pointer Lock, V. Scheib. W3C.


Preload, I. Grigorik. W3C.

Presentation API, M. Foltz, D. Rötsches. W3C.

Referrer Policy, J. Eisinger, E. Stark. W3C.

Cooperative Scheduling of Background Tasks, R. McIlroy, I. Grigorik. W3C.

Resource Hints, I. Grigorik. W3C.


Requirements for Internet Hosts -- Application and Support, R. Braden. IETF, October 1989.

Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, N. Freed, N. Borenstein. IETF.

The "data" URL scheme, L. Masinter. IETF.

Internet Calendaring and Scheduling Core Object Specification (iCalendar), B. Desruisseaux. IETF.

URL Resolution Services Necessary for URN Resolution, M. Mealling, R. Daniel. IETF.

The Text/Plain Format and DelSp Parameters, R. Gellens. IETF.

Registration Procedures for Message Header Fields, G. Klyne, M. Nottingham, J. Mogul. IETF.

Scripting Media Types, B. Höhrmann. IETF.
The WebSocket protocol, I. Fette, A. Melnikov. IETF.


XFN 1.1 profile, T. Çelik, M. Mullenweg, E. Meyer. GMPG.

XMLHttpRequest, A. van Kesteren. WHATWG.

(XNon-normative) Substitutions, Randall Munroe. xkcd.

Extensible Markup Language, T. Bray, J. Paoli, C. Sperber-McQueen, E. Maler, F. Yergeau. W3C.

(XML) Entity Definitions for Characters, D. Carlisle, P. Ion. W3C.

Namespaces in XML, T. Bray, D. Hollander, A. Layman, R. Tobin. W3C.

Associating Style Sheets with XML documents, J. Clark, S. Pieters, H. Thompson. W3C.

XML Path Language (XPath) Version 1.0, J. Clark, S. DeRose. W3C.

XSL Transformations (XSLT) Version 1.0, J. Clark. W3C.

DOM XSLTProcessor, WHATWG Wiki. WHATWG.
Acknowledgments

Thanks to Tim Berners-Lee for inventing HTML, without which none of this would exist.

Vladimir Vukićević, Vyacheslav Aristov, voracity, Walter Steiner, Wakaba, Wayne Carr, Wayne Pollock, Wellington Fernando de Macedo, Weston Ruter, Wilhelm Joys Andersen, Will Levine, Will Ray, William Chen, William Swanson, Willy Martin Aguirre Rodríguez, Vladmir Palant, Wojciech Mach, Wolfram Krieseing, Xan Gregg, xenotheme, XhmikosR, Xida Chen, Xidorn Quan, Xue Fuqiao, Yang Chen, Yao Xiao, Yash Handa, Yay295, Ye-Kui Wang, Yehuda Katz, Yi Xu, Yi-An Huang, Yngve Nysaeter Pettersen, Yoav Weiss, Yonathan Randolph, Yu Huojian, Yury Delendik, 平野裕 (Yutaka Hirano), Yuzo Fujishima, 張智強 (Zhiqiang Zhang), 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.

Special thanks to Richard Williamson for creating the first implementation of canvas in Safari, from which the canvas feature was designed.

Special thanks also to the Microsoft employees who first implemented the event-based drag-and-drop mechanism, contenteditable, and other features first widely deployed by the Windows Internet Explorer browser.

Special thanks and $10,000 to David Hyatt who came up with a broken implementation of the adoption agency algorithm that the editor had to reverse engineer and fix before using it in the parsing section.

Thanks to the participants of the microdata usability study for allowing us to use their mistakes as a guide for designing the microdata feature.

Thanks to the many sources that provided inspiration for the examples used in the specification.

Thanks also to the Microsoft blogging community for some ideas, to the attendees of the W3C Workshop on Web Applications and Compound Documents for inspiration, to the #mrt crew, the #mrt.no crew, and the #whatwg crew, and to Pillar and Hedral for their ideas and support.

Thanks to Igor Zhbanov for generating PDF versions of the specification.

Special thanks to the RICG for developing the picture element and related features; in particular thanks to Adrian Bateman, Bruce Lawson, David Newton, Ilya Grigorik, John Schoenick, Leon de Rijke, Mat Marquis, Marcos Cáceres, Tab Atkins, Theresa O'Connor, and Yoav Weiss for their contributions.

Special thanks to the WPWG for incubating the custom elements feature. In particular, thanks to David Hyatt and Ian Hickson for their influence through the XBL specifications, Dimitri Glazkov for the first draft of the custom elements specification, and to Alex Komoroske, Andres Rios, Boris Zbarsky, Brian Kardell, Daniel Buchner, Dominic Cooney, Erik Arvidsson, Elliott Sprehn, Hajime Morita, Hayato Ito, Jan Miksovsky, Jonas Sicking, Olli Pettay, Rafael Weinstein, Roland Steiner, Ryosuke Niwa, Scott Miles, Steve Faulkner, Steve Orvell, Tab Atkins, Theresa O'Connor, Tim Perry, and William Chen for their contributions.

Special thanks to the CSSWG for developing the worklets specification. In particular, thanks to Ian Kilpatrik for his work as editor of the original worklets specification.

For about ten years starting in 2003, this standard was almost entirely written by Ian Hickson (Google, ian@hixie.ch). More recently, Simon Pieters (Bocoup, zcorpan@gmail.com), Anne van Kesteren (Mozilla, annievk@annevk.nl), Philip Jägenstedt (Google, philip@foolip.org), and Domenic Denicola (Google, d@domenic.me), all previously long-time contributors, have joined Ian in editing the text directly.
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'Neil. (Public domain)

The image of the kettlebell swing in the embedded content introduction is based on a photo by kokkarina. (CC0 1.0)

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)

The diagram showing how `async` and `defer` impact `script` loading is based on a similar diagram from a blog post by Peter Beverloo. (CC0 1.0)

The image decoding demo used to demonstrate module-based workers draws on some example code from a tutorial by Ilmari Heikkinen. (CC BY 3.0)

The `<flag-icon>` example was inspired by a custom element by Steven Skelton. (MIT)

Part of the revision history of the `picture` element and related features can be found in the ResponsiveImagesCG/picture-element repository, which is available under the W3C Software and Document License.

Part of the revision history of the `theme-color` metadata name can be found in the whatwg/meta-theme-color repository, which is available under CC0.

Part of the revision history of the `custom elements` feature can be found in the w3c/webcomponents repository, which is available under the W3C Software and Document License.

Part of the revision history of the `innerText` getter and setter can be found in the rocallahan/innerText-spec repository, which is available under CC0.

Part of the revision history of the `worklets` feature can be found in the w3c/css-houdini-drafts repository, which is available under the W3C Software and Document License.

Copyright © WHATWG (Apple, Google, Mozilla, Microsoft). This work is licensed under a Creative Commons Attribution 4.0 International License.

This is the Living Standard. Those interested in the patent-review version should view the Living Standard Review Draft.